UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

FACULTAD DE INGENIERÍA ELECTRÓNICA Y ELÉCTRICA

E.A.P. DE INGENIERÍA ELECTRÓNICA

Diseño e implementación de un robot móvil con control de trayectoria mediante principios odométricos

TESIS

Para optar el Título Profesional de Ingeniero Electrónico

AUTOR

Luis Alberto Arellano Zea

ASESOR

Bruno Elio Vargas Tamani

Lima – Perú

2015

Dedico mi tesis a mis padres quienes me apoyaron incondicionalmente durante toda mi formación académica y personal.

Luis Alberto Arellano Zea

"Lo que conduce y mueve al mundo no son las máquinas sino las ideas".

Víctor Hugo.

Agradecimientos

En primer lugar quisiera agradecer a mi familia por todo el apoyo brindado durante la elaboración del proyecto.

De igual manera, agradezco a mi asesor Dr. Bruno E. Vargas Tamani quien me brindó su ayuda y consejos desde mis primeros años en la facultad hasta culminar mi tesis.

Mi agradecimiento a Daniel Gustavo Martinez Martinez, con quien desarrollé el proyecto desde los inicios, obteniendo muy buenos resultados.

Así mismo, agradezco a Jorge Abraham Salgado Hervias por haberme guiado en el mundo de la robótica.

Por último mi agradecimiento especial a mi Alma Mater, La Universidad Nacional Mayor de San Marcos por tantas experiencias vividas que fortalecieron mi formación como profesional.

Índice General

Resumen

El presente trabajo de tesis consiste en el diseño e implementación de un robot móvil de tres grados de libertad, capaz de controlar su posición y trayectoria en un plano cartesiano, además de posicionarse en lugares definidos por el usuario. El objetivo del proyecto es controlar el movimiento del robot, manipulando su traslación y rotación de manera precisa y eficiente.

El móvil utiliza dos motores acoplados a llantas para su locomoción, estos motores están colocados en una configuración diferencial, haciendo que el desplazamiento y la rotación sobre su eje sea mucho más eficiente. El robot cuenta con un sistema de medición basado en dos encoders incrementales situados a los lados de los motores. Las señales generadas por estos sensores son procesadas por el móvil, el cual hará el análisis cinemático en línea empleando principios de odometría y ecuaciones en diferencia para estimar la posición y orientación relativa del robot. El resultado de esta operación es utilizado en el algoritmo de control, que consiste en dos controladores PID (proporcional, integral y derivativo) discretos [1]. El primero controla la orientación del robot, asegurando que se posicione en el ángulo correcto antes de iniciar su movimiento y durante el recorrido lineal para que el móvil no se desvíe de su trayectoria. El segundo controlador PID regula la posición lineal del robot en función de las coordenadas iniciales y finales de la trayectoria trazada. Este recorrido es planificado en línea en función a las coordenadas de puntos predefinidos en la lógica de generación de trayectorias.

El robot es monitoreado en tiempo real por una computadora que a través de una interfaz gráfica desarrollada en Java permite observar los parámetros de control en cuadros de texto y gráficas dinámicas. Además, permite el envío de comandos pre configurados y secuencias de trayectorias lineales. Para establecer la conexión entre el robot y la PC se utilizó comunicación serial asíncrona bajo el estándar RS-232 y utilizando el protocolo UART. La unidad de procesamiento para la implementación de lógica y algoritmos de control fue un dsPIC30F4011 [2] (controlador digital de señales), ya que posee una alta velocidad para el procesamiento de señales y operaciones matemáticas de punto flotante. Además, cuenta con módulos especializados para el

control de motores y comunicación serial, haciendo que la programación sea mucho más eficiente.

Al finalizar la implementación del robot, este mostró muy buenos resultados durante las pruebas cumpliendo con los algoritmos de control de rotación y traslación, así como el monitoreo y control desde la PC. Uno de los principales aportes de este trabajo es que se demostró poder tener un control eficiente y preciso de un robot móvil empleando únicamente 2 encoders como sistema de medición.

Abstract

The present thesis consists in the design and implementation of a mobile robot of three degrees of freedom, able to control their position and trajectory in a Cartesian plane, besides being positioned in user-defined locations. The objective of the project is controlling the movement of the robot, manipulating its translation and rotation accurately and efficiently.

The robot uses two motors coupled wheels for locomotion, these engines are placed in a differential configuration, causing the displacement and rotation on its axis much more efficient. The robot has a measurement system based on two incremental encoders situated on the sides of the engines. The signals generated by these sensors are processed by the robot, which will do a kinematic analysis in line using odometry principles and difference equations to estimate the relative position and orientation of the robot. The result of this operation is used in the control algorithm, which consists of two discrete PID controllers (proportional, integral and derivative). The first controls the orientation of the robot, ensuring that it is positioned at the correct angle before starting its motion and during the linear path in order to the robot does not deviate from its trajectory. The second linear PID controller regulates the position of the robot according to the initial and final coordinates of the traced path. This trajectory is planned in line according to the coordinates of the predefined points in the logic of paths generation.

The robot is monitored in real time by a computer through a graphical interface developed in Java, which allows observing the control parameters in dynamic text boxes and graphics. Additionally, allows sending pre-configured commands and sequences of linear trajectories. To establish the connection between the robot and the PC, it has used serial asynchronous communication under the RS- 232 standard and using the UART protocol. The processing unit for the implementation of logic and control algorithms was a dsPIC30F4011 (digital signal controller), as it has a high-speed signal processing and floating point math operations. It also has specialist modules for motor control and serial communication, making programming much more efficient.

After the implementation of the robot, this showed very good results during testing, compliance with the rotation and translation control algorithms, as well as monitoring and controlling from the PC. One of the main contributions of this work is that it showed that you could have an efficient and accurate control of a mobile robot with three degrees of freedom using only two encoders as a measurement system.

Introducción

La robótica es una disciplina en la cual convergen muchas ramas de la ingeniería como la electrónica, sistemas, control, mecánica, etc. Debido a la complejidad de los sistemas robotizados, especialmente en los robots autónomos no tripulados. Cada una de estas disciplinas es fundamental para el desarrollo del proyecto. En la presente tesis se tomó especial énfasis en las áreas de electrónica, sistemas y control.

El problema que se tratará en este proyecto es el sistema de control y la navegación de un robot móvil autónomo de tres grados de libertad, debido a la gran inserción de los robots móviles en diferentes sectores de la sociedad como la industria, minería, defensa, etc., para tareas de exploración, transporte y recolección de productos. Dichas tareas requieren que el robot posea una unidad de procesamiento de datos de buena capacidad y velocidad de respuesta elevada, destinados a la adquisición de las señales de sensado de posición, controladores discretos y algoritmos de control implementados en el sistema embebido. Además debe poseer un sistema robusto capaz de controlar su trayectoria durante la navegación, así como un sistema de monitoreo, supervisión y control en tiempo real.

El objetivo central del proyecto es desarrollar un robot móvil con un sistema de control preciso y eficiente, que sea de plataforma abierta para el posterior desarrollo de aplicaciones específicas. Para poder cumplir con dicho objetivo se implementó el robot por completo, lo que conllevó a diseñar la electrónica y mecánica previo al desarrollo de los sistemas de control embebidos en el microcontrolador.

Se implementó una estación de monitoreo, supervisión y control de los parámetros de movimiento, configuración y estados del robot. Dicha aplicación fue desarrollada bajo la plataforma de Java, en la cual se observará todas las variables controladas en gráficos y cuadros de texto así como una sección para enviar las coordenadas en las cuales se posicionará el robot.

Descripción del sistema

El sistema es un robot móvil que se desplaza sobre un plano libre de obstáculos y discontinuidades, el cual emplea ruedas acopladas a dos motores como sistema de locomoción tal como se observa en la figura intro-1.

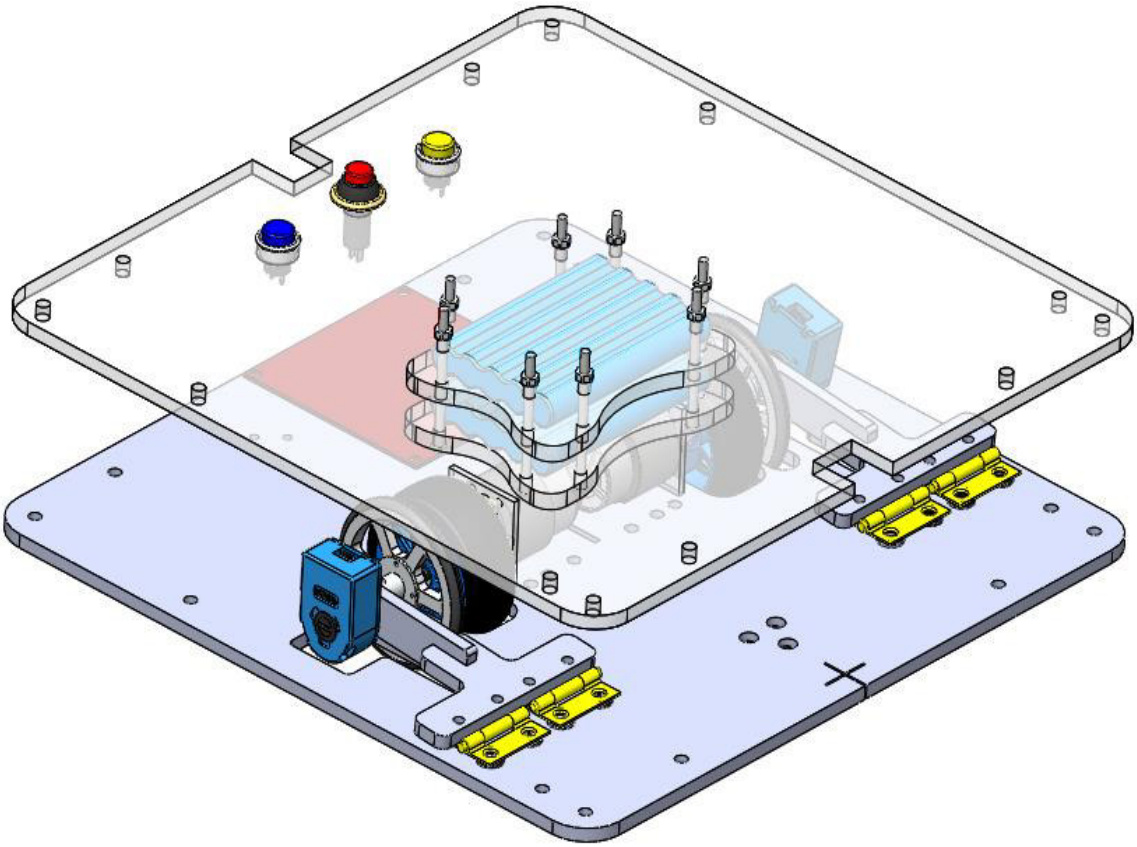


Figura Intro-1.-Vista del Robot.

El robot es un sistema autónomo gobernado por un controlador embebido que procesa las señales provenientes de los sensores de posición (encoders) para estimar los parámetros de movimiento: las coordenadas (x,y) y el ángulo de orientación (θ) respecto al eje X tal como se muestra en la figura intro-2. Estos parámetros son empleados posteriormente para controlar el movimiento del robot durante su desplazamiento.

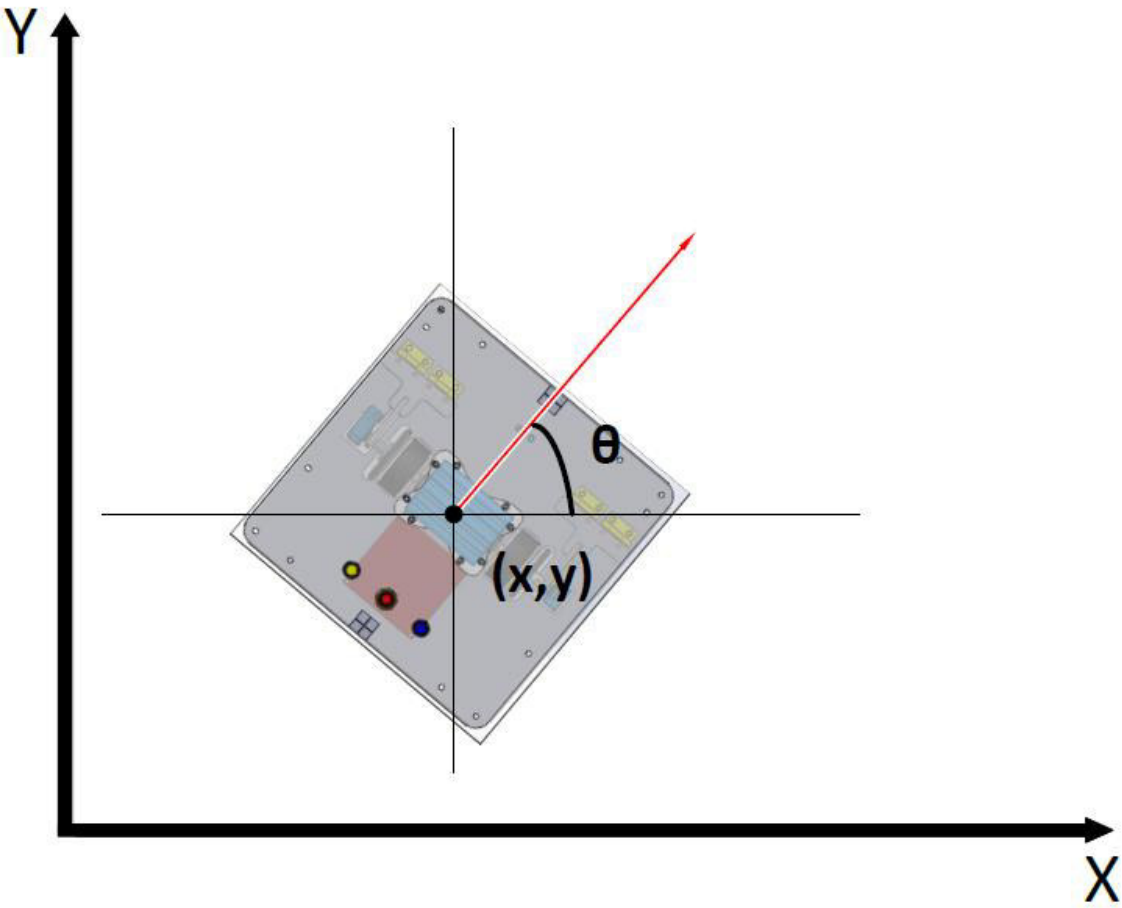


Figura intro-2.-Parámetros del Robot.

El robot se desplaza entre dos puntos siguiendo una trayectoria lineal recta, como se observa en la figura intro-3, el robot se desplaza del punto A al punto B. Este movimiento es controlado para disminuir el error de posición generado durante el desplazamiento, asegurando que el móvil se traslade de un punto a otro con un error mínimo aceptable.

Extrapolando el principio de desplazamiento entre dos puntos el robot es capaz de recorrer una serie definida de puntos de manera secuencial. Si observamos la figura intro-4, el robot se desplaza del punto A al punto B, una vez culminado este recorrido se desplaza del punto B al punto C y así sucesivamente. De esta manera el sistema es capaz de movilizarse por rutas complejas descomponiendo su recorrido en segmentos de recta.

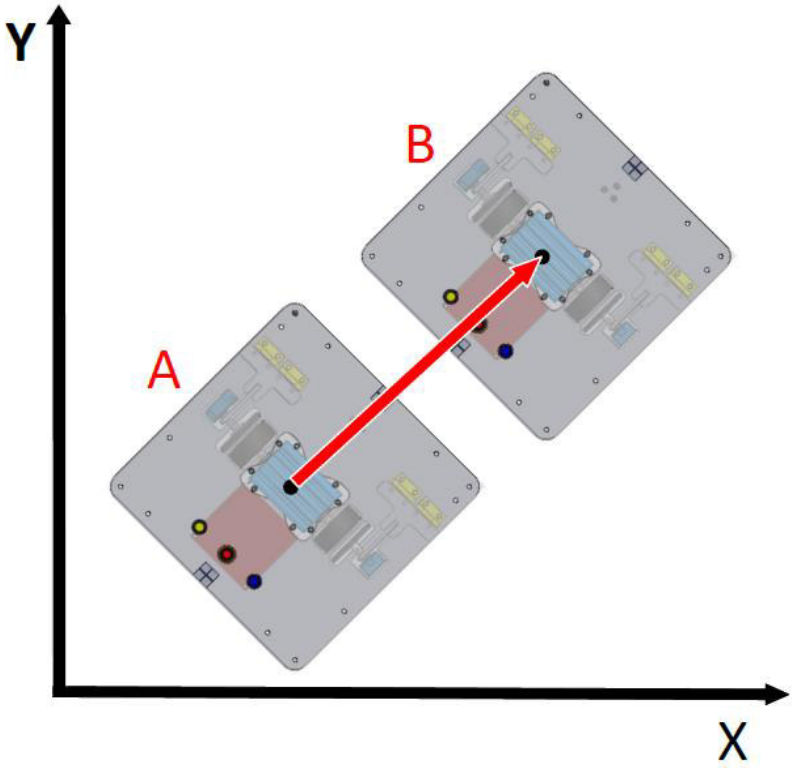


Figura intro-3. Trayectoria lineal del robot

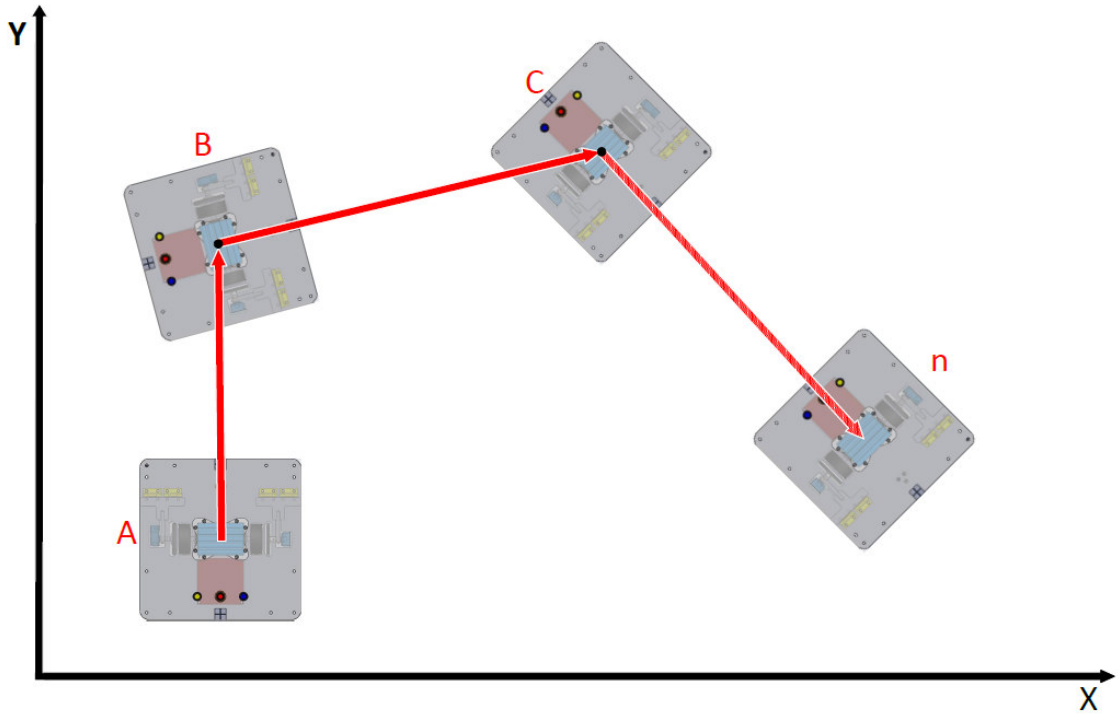


Figura intro-4.-Trayectoria secuencial de robot

El robot recibe las coordenadas (x,y) de la secuencia de puntos en los cuales debe posicionarse desde una PC (ver figura intro-5) empleando una comunicación serial entre ambos dispositivos. Esta PC contiene un programa informático desarrollado en Java dedicado para la supervisión y monitoreo del sistema. Su función es proporcionar las consignas de posicionamiento de acuerdo a la ruta trazada por el operador del robot, así como mostrar de manera gráfica el recorrido del móvil, coordenadas (x,y), la orientación angular (θ), velocidad, etc.

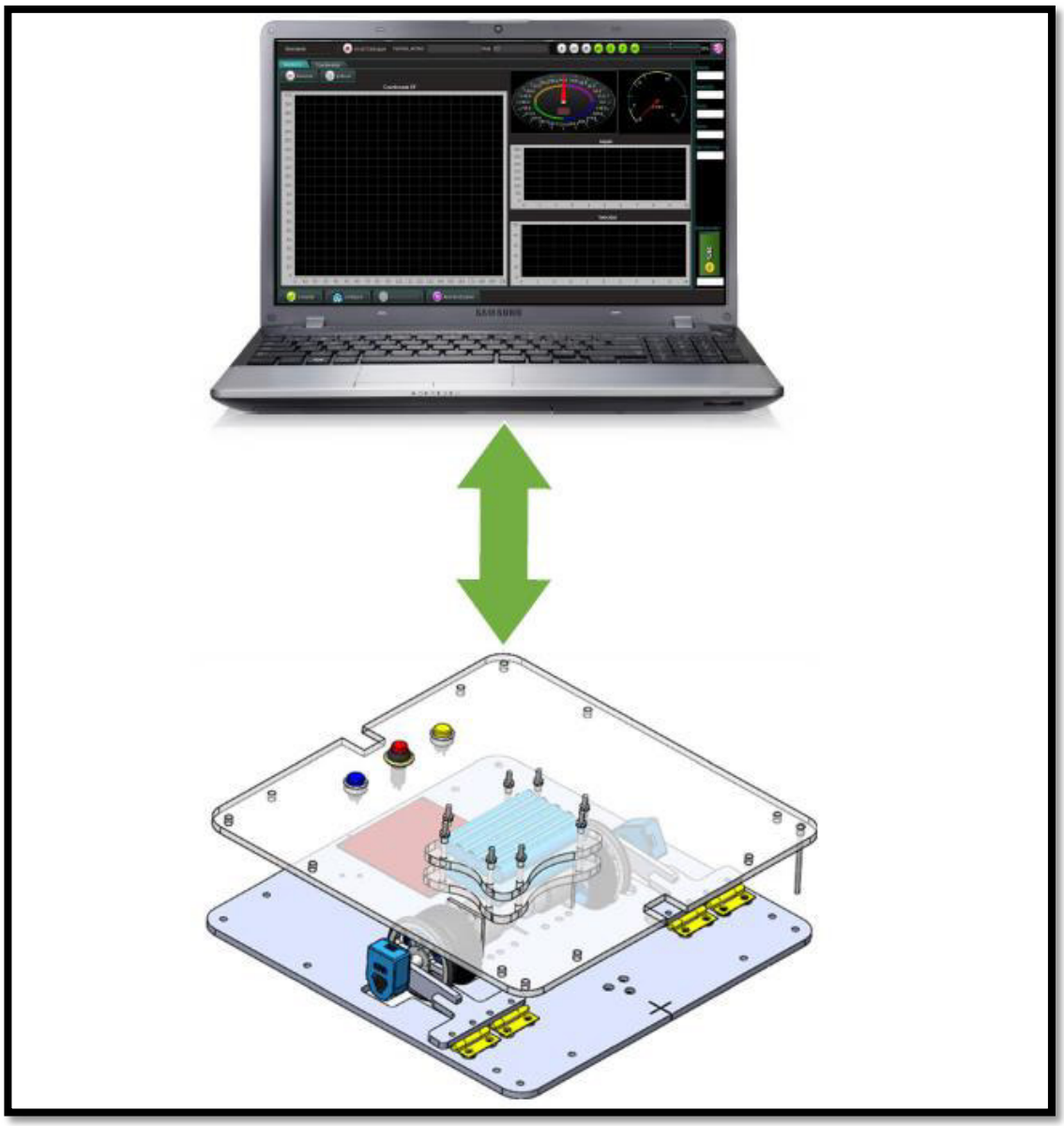


Figura intro-5.-Interacción PC - Robot

Capítulo I
Marco teórico

1. Descripción del robot móvil

La robótica móvil se ha desarrollado en conjunto con los avances tecnológicos recientes, lo cual ha generado el desarrollo de muchas aplicaciones en la sociedad. Los robots hoy en día son en su mayoría excesivamente costosos y su función está muy especializada, por lo que los estudios actuales se enfocan en conseguir que sean de aplicaciones generales y más económicas [3].

Los robots se clasifican de la siguiente manera en función de su aplicación y características.

- Robot móvil
- Robot industrial
- Robot humanoide
- Robot inteligente
- Robot de servicios
- Robot híbrido

Este trabajo se centra en el estudio de los robots móviles con ruedas, en especial de los microbots, los cuales están destinados a realizar pequeñas tareas con rapidez y precisión, simular actividades de humanos como la recolección y exploración. Su principal filosofía es la movilidad, lo cual es muy ventajoso al momento de realizar determinadas tareas y poder trabajar en equipo de manera colaborativa o de manera aislada. Son útiles para realizar trabajos en lugares peligrosos y de difícil acceso por el hombre.

El comportamiento y la forma de actuar de los robots móviles está en función del programa que se ejecuta sobre él. El robot realizará autónomamente las tareas de modo eficiente en función de la construcción mecánica, diseño electrónico, sistemas de control y programación. La precisión en su desplazamiento está en función del sistema mecánico y del diseño electrónico, mientras que la autonomía y la inteligencia se encuentran definidos por el programa que gobierna el robot. El crecimiento de la

autonomía abre la puerta a nuevas aplicaciones comerciales. Sin embargo la autonomía es compleja, y salvo casos contados, el desarrollo de aplicaciones con robots sigue siendo un tema de investigación.

Los robots móviles por lo general están destinados a simular el compartimiento de personas y animales con un nivel de eficiencia similar, la estructura de un robot móvil es muy similar a las de un ser vivo. Ambos cuentan con un sistema estructural que soporta los demás subestructuras. Poseen un sistema de sensoramiento y de locomoción; así como un sistema de procesamiento de información para la toma de decisiones. Si analizamos los subsistemas del robot móvil encontraremos lo siguiente:

- Estructura mecánica: Compuesta por un armazón sólido generalmente hecho de algún metal donde se colocan todos los componentes del robot.

- Sensores: Elementos que proporcionan información al sistema de control para la realimentación. Pueden ser de tipo digitales o analógicos; por ejemplo encoders, sonares, láser, cámaras, etc.

- Actuadores: Es el elemento final de control, el cual permite que el móvil interactúe con el medio, generando la locomoción para el desplazamiento del robot. El elemento más usado para este fin es el motor eléctrico.

- Unidad de procesamiento: En este sistema se implementan los algoritmos de control y navegación mediante la programación de un dispositivo embebido que utilizará los sensores como entradas y los actuadores como salidas. Esto definirá el comportamiento del robot, así como sus características de precisión y fidelidad en su desplazamiento.

El primer paso en la construcción de los robots móviles es la elección de su configuración, esto es, definir como estarán distribuidos los principales elementos que lo componen: ruedas, motores y encoders [4]. El robot móvil desarrollado en la presente tesis posee la configuración diferencial. Esta cuenta con dos ruedas situadas diametralmente opuestas en un eje perpendicular a la dirección del robot, tal como se observa en la figura 1.1.

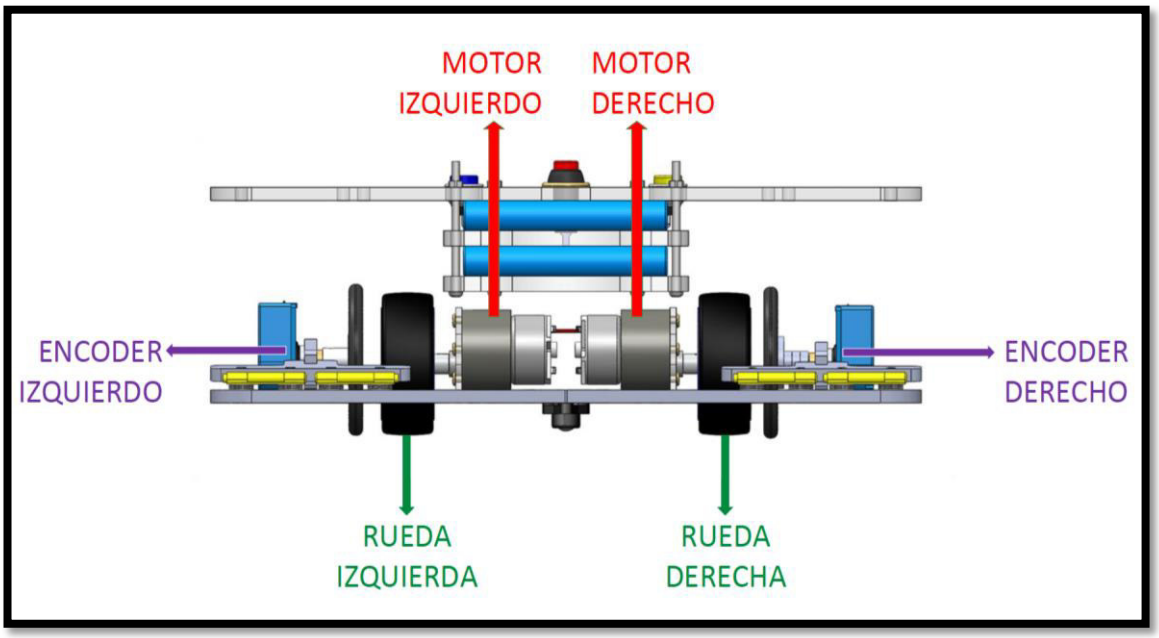


Figura 1.1.-Configuracion diferencial del robot.

Cada una de las ruedas irá acoplada a un motor independiente, por lo que podrán tener velocidades distintas, generando un ángulo de orientación (θ), tal como se muestra en la figura 1.2 . Este ángulo está en función de las velocidades del motor del lado izquierdo (Vi) y del lado derecho (Vd). Dichas velocidades generan una fuerza resultante (Fy), que posee la misma dirección de desplazamiento lineal del robot.

Para que el robot se desplace hacia adelante las velocidades de ambos motores deberán ser iguales escalarmente y tener el mismo sentido. Si la velocidad del motor del lado derecho (Vd) es mayor a la velocidad del motor del lado izquierdo (Vi), el robot se desplazará hacia la izquierda. Si la velocidad del motor del lado izquierdo (Vi) es mayor a la velocidad del motor del lado derecho (Vd), el robot se desplazará hacia la derecha.

Una de las ventajas de la configuración diferencial radica en que el robot puede girar sobre su propio eje en sentido horario o anti horario (ver figura 1.3). Para realizarlo, las velocidades de los motores Vd y Vi tienen que ser iguales escalarmente, pero de sentidos opuestos. Esta característica se empleará para cambiar la orientación del robot móvil sin variar sus coordenadas (x,y).

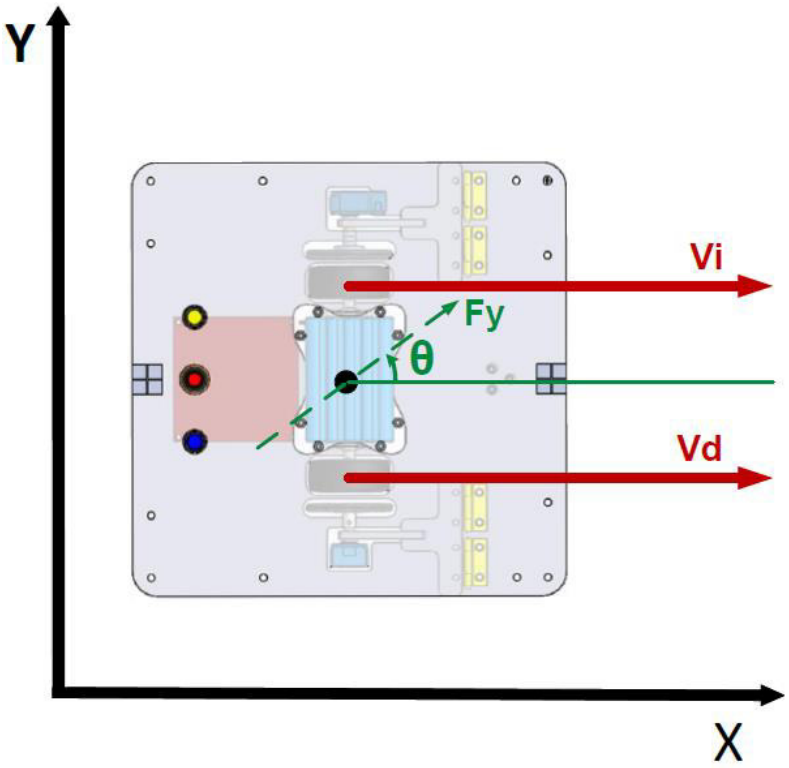


Figura 1.2.-Desplazamiento del robot en configuración diferencial.

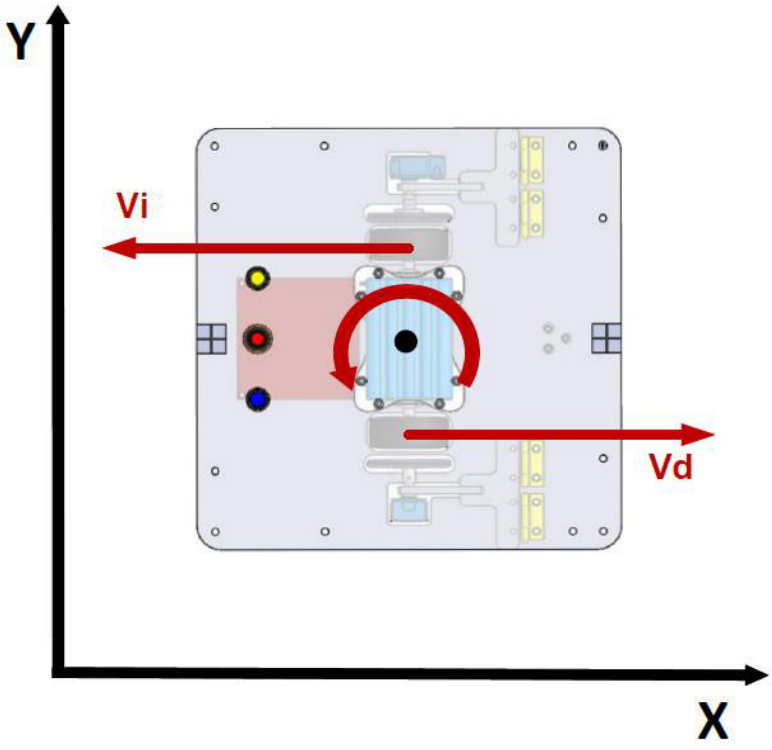


Figura 1.3.-Rotación del robot sobre su eje.

2. Encoder

Los encoders son mecanismos utilizados para calcular la posición, velocidad y aceleración de un eje [5]. Este dispositivo optoelectrónico convierte el movimiento angular de su eje en un tren de pulsos digitales. Su principio de funcionamiento consiste en la detección de un haz generado por un emisor de luz que atraviesa las ranuras de un disco a través de un fotoreceptor, tal como se muestran en las figuras 1.4 y 1.5.

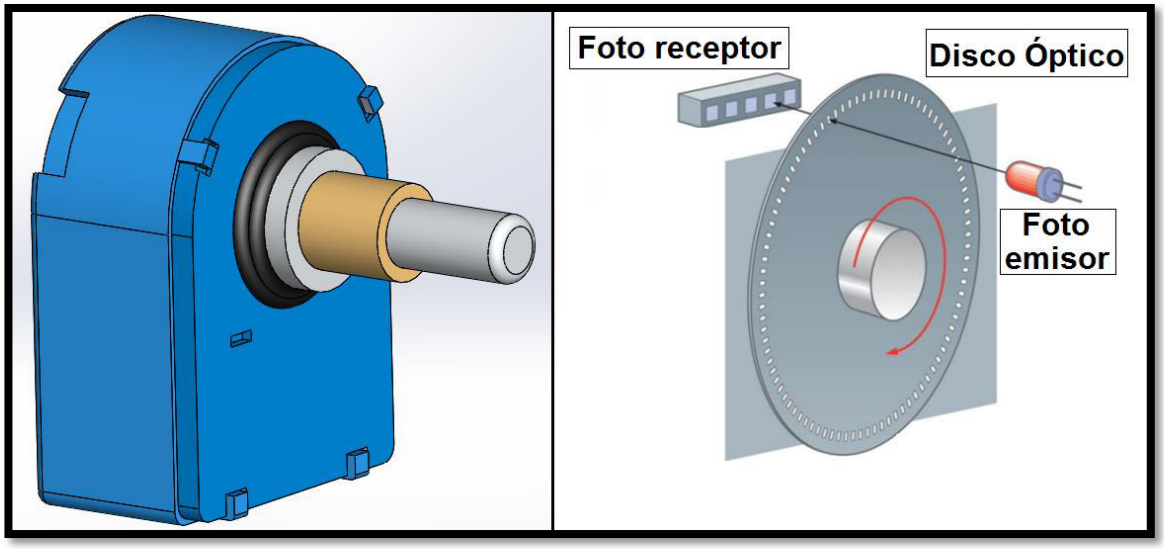


Figura 1.4.-Esquema de un encoder – vista 1.

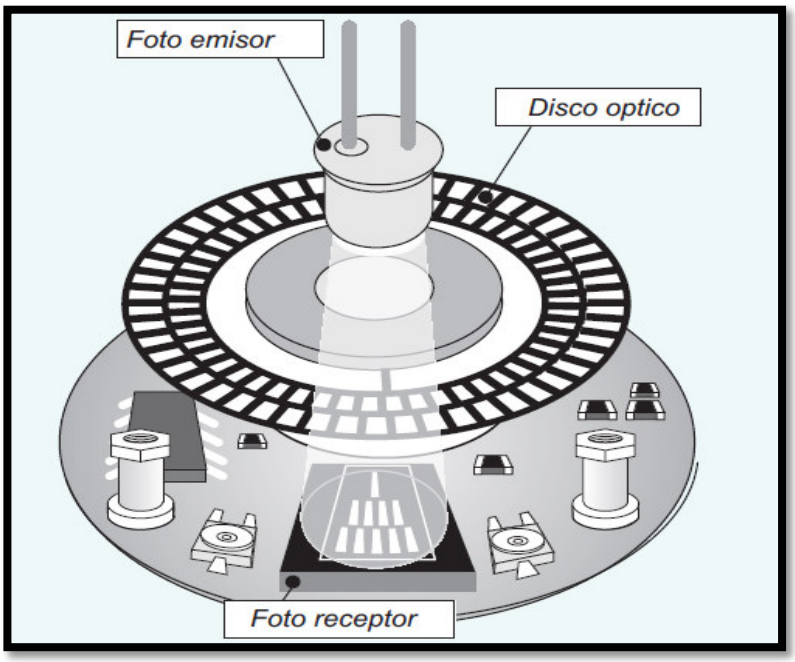


Figura 1.5.-Esquema de un encoder – vista 2.

Cuando el fotoreceptor detecta el haz de luz se obtiene un "1" lógico y cuando no se detecta se obtiene un "0" lógico. De esta manera el encoder generará un tren de pulsos en función al movimiento angular transmitido a su eje.

Los encoders proporcionan dos ondas cuadradas y desfasadas entre sí en 90º, los cuales llamaremos canal “A” y canal “B”, según se muestra en la figura 1.6. Con la lectura de un solo canal (canal “A”) se dispone de la información necesaria para calcular únicamente la velocidad de rotación, mientras que si se capta también la señal “B” es posible discriminar el sentido de rotación en base a la secuencia de datos que producen ambas señales.

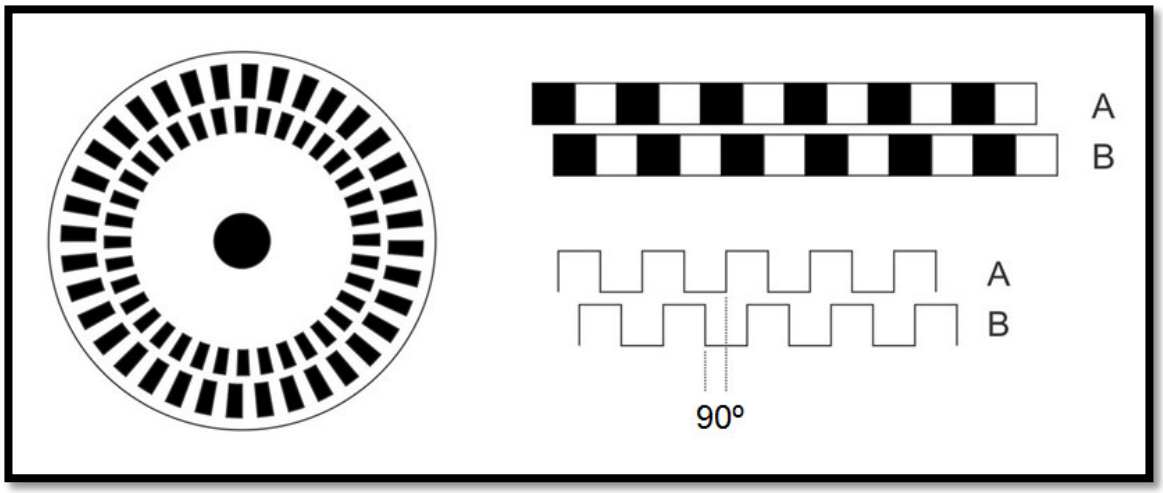


Figura 1.6.- Tren de pulsos generados por los canales A y B del encoder.

Si agrupamos las dos señales generadas por ambos canales se obtiene un código binario (AB) con el cual es posible determinar la posición angular y el sentido de rotación del eje del encoder. En la figura 1.7 se muestra el código generado cuando el encoder de desplaza en sentido horario y anti-horario. En el primer caso muestra la secuencia de códigos generados al desplazar el eje durante un ciclo del pulso generado por el canal “A” o “B”. A partir de ahora llamaremos a este ciclo como “Tick”, el cual será identificado a partir de la secuencia de códigos binarios, por ejemplo 01, 00, 10 11. Esta secuencia será un “Tick” positivo ya que el eje rotó en sentido horario. Si observamos ahora el segundo caso en el cual el eje del encoder rotó en sentido anti-horario, la secuencia de códigos binarios (11, 10, 00, 01) es opuesta a la secuencia anterior, por lo que será un “Tick” negativo.

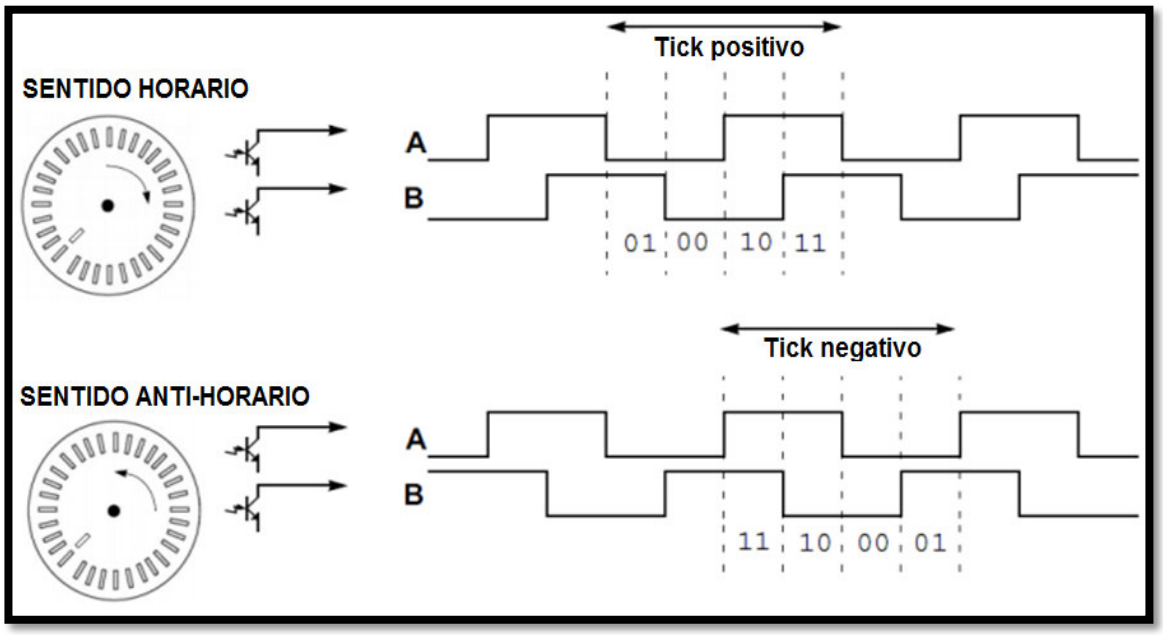


Figura 1.7.- Generación de Tick positivo y negativo.

Para medir el desplazamiento tangencial asociado al eje del encoder se acoplará una rueda tal como se muestra en la figura 1.8. De tal manera que el movimiento producido por la rueda se transmitirá al eje del encoder generando Ticks positivos y negativos en función al sentido de giro.

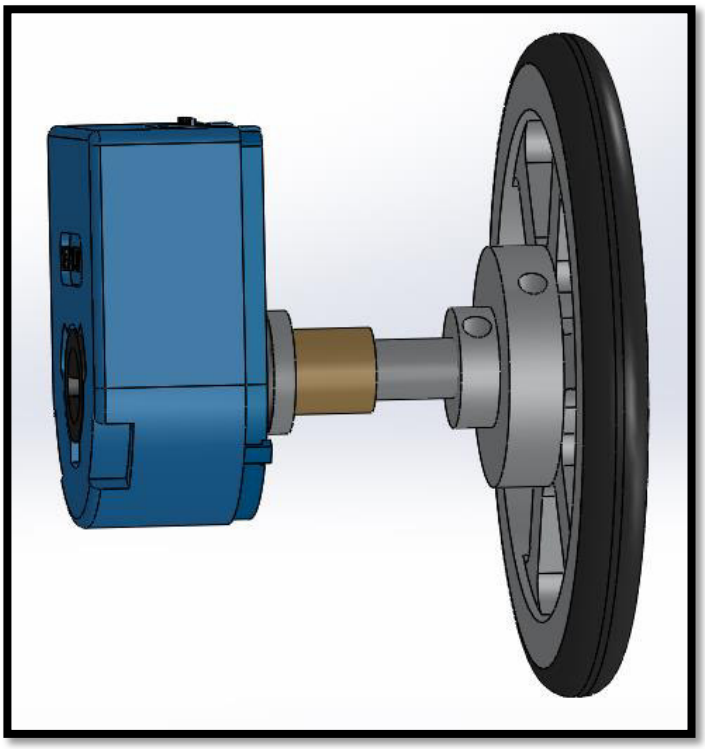


Figura 1.8.- Acoplamiento entre encoder y rueda.

En la figura 1.9 se muestra el desplazamiento de la rueda unida al eje del encoder. Dicha rueda realiza un movimiento lineal (L), el cual se transmite al eje del encoder como una rotación angular (α). Este desplazamiento generará cierta cantidad de Ticks (T1), los cuales se emplearán para calcular dicha rotación angular (α) en la ecuación 1.1 y posteriormente el desplazamiento lineal (L) en la ecuación 1.2.

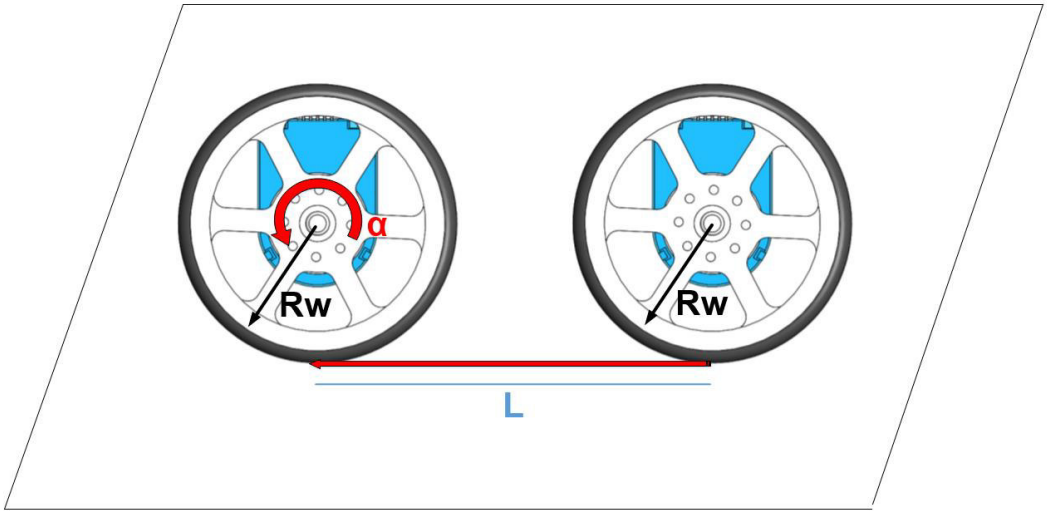


Figura 1.9.- Acoplamiento entre encoder y rueda.

$$\alpha = \frac{T_1}{T_R} \times 2\pi \ rad \ \ldots (1.1)$$

Donde:

α	=	Ángulo de rotación de la rueda (rad).
T_1	=	Cantidad de ticks generados por el encoder.
T_R	=	Cantidad de ticks en una revolución.

A partir de esta ecuación podemos calcular el ángulo mínimo que podrá detectar el encoder, considerando que este posee 500 ticks en una revolución.

$$\frac{1}{500} \times 2\pi \ = 0.01256 \ rad \ = 0.72°$$

Una vez encontrado el ángulo de rotación se calculará la distancia lineal recorrida por la rueda, el cual será igual al arco generado por el ángulo (α).

$$\boldsymbol{L = \alpha \times Rw} \quad \ldots (\mathbf{1.2})$$

Donde:

L = Distancia lineal recorrida por la rueda (m).

α = Ángulo de rotación de la rueda (rad).

Rw = Radio de la rueda (m).

Reemplazando la ecuación 1.1 en 1.2 se obtiene la distancia lineal en función de la cantidad de Ticks generados a partir de dicho desplazamiento (ecuación 1.3).

$$\boldsymbol{L = \frac{T_1}{T_R} \times 2\pi \times Rw} \quad \ldots (\mathbf{1.3})$$

De igual manera al caso anterior partir de esta ecuación podemos calcular la resolución o la mínima distancia lineal recorrida por la rueda que podrá detectar el encoder, considerando que este posee 500 ticks en una revolución y esta acoplado a una rueda de 0.035 metros de radio.

$$\boldsymbol{\frac{1}{500} \times 2\pi \times 0.035 = 0.000439823\ m = 0.439823\ mm}$$

3. Odometría

La odometría es la estimación de la posición (coordenadas (x,y) y ángulo de orientación θ) de vehículos con ruedas durante su navegación respecto a un punto o coordenada inicial, tal como se muestra en la figura 1.10. Para realizar esta estimación se emplea la información obtenida a través de sensores que miden rotación de las ruedas del móvil durante el desplazamiento, siendo los encoders [6] uno de los más utilizados debido a su resolución (ver Capítulo 1, 2.Encoder). Los encoders se colocan en los extremos del robot cuando se emplea la configuración diferencial para la locomoción de sus motores (ver Capítulo 1, 1.Descripción del robot móvil).

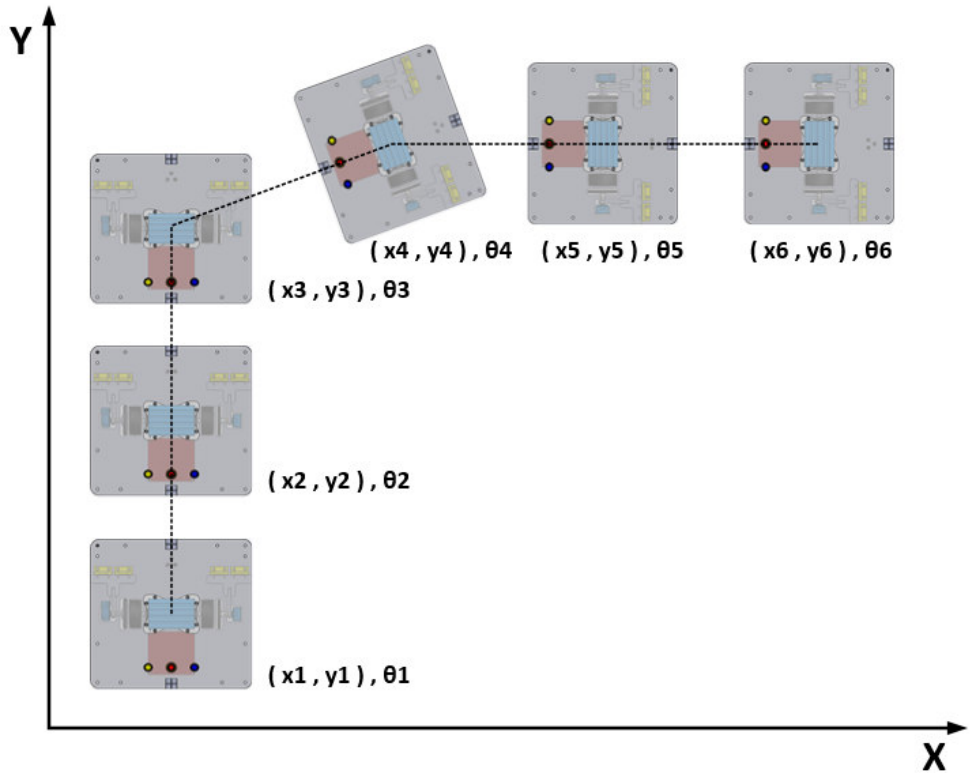


Figura 1.10.- Desplazamiento odométrico del robot.

La odometría está basada en la suposición que las revoluciones de las ruedas que están acopladas a los encoders traduzcan el desplazamiento real del robot sobre el suelo. Para asegurar esta condición no se acoplarán las ruedas de los motores directamente al eje encoder, ya que esto generaría falsas lecturas si las ruedas de los motores patinasen sobre la superficie del suelo [7]. Se optó por colocar ruedas independientes acopladas al eje de los encoders, tal como se muestra en la figura 1.11, donde cada motor tiene su propia rueda para generar el desplazamiento del robot y cada encoder también posee su rueda para medir el desplazamiento generado.

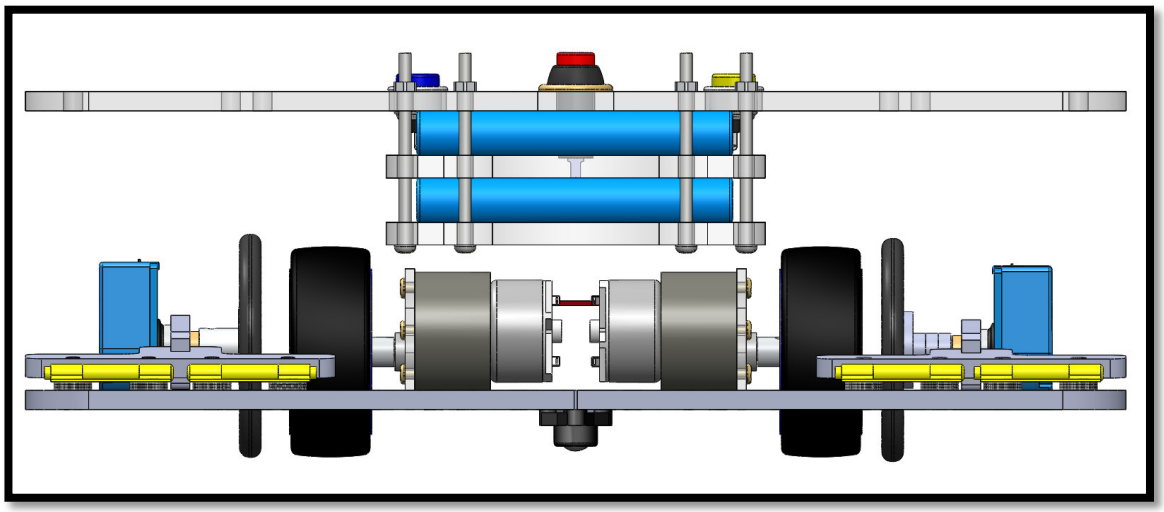


Figura 1.11.- Ruedas independientes de encoders.

Las ruedas que están acopladas a los encoders poseen un mecanismo que les permite poder desplazarse hacia arriba y abajo. Este movimiento, el cual se muestra en la figura 1.12, facilita que la rueda del encoder se pueda adaptar fácilmente a la superficie del suelo, generando mayor contacto entre ambos durante el desplazamiento. Este mecanismo consiste en una pieza cuyos extremos están unidos al encoder y a dos bisagras, los cuales permiten el libre movimiento.

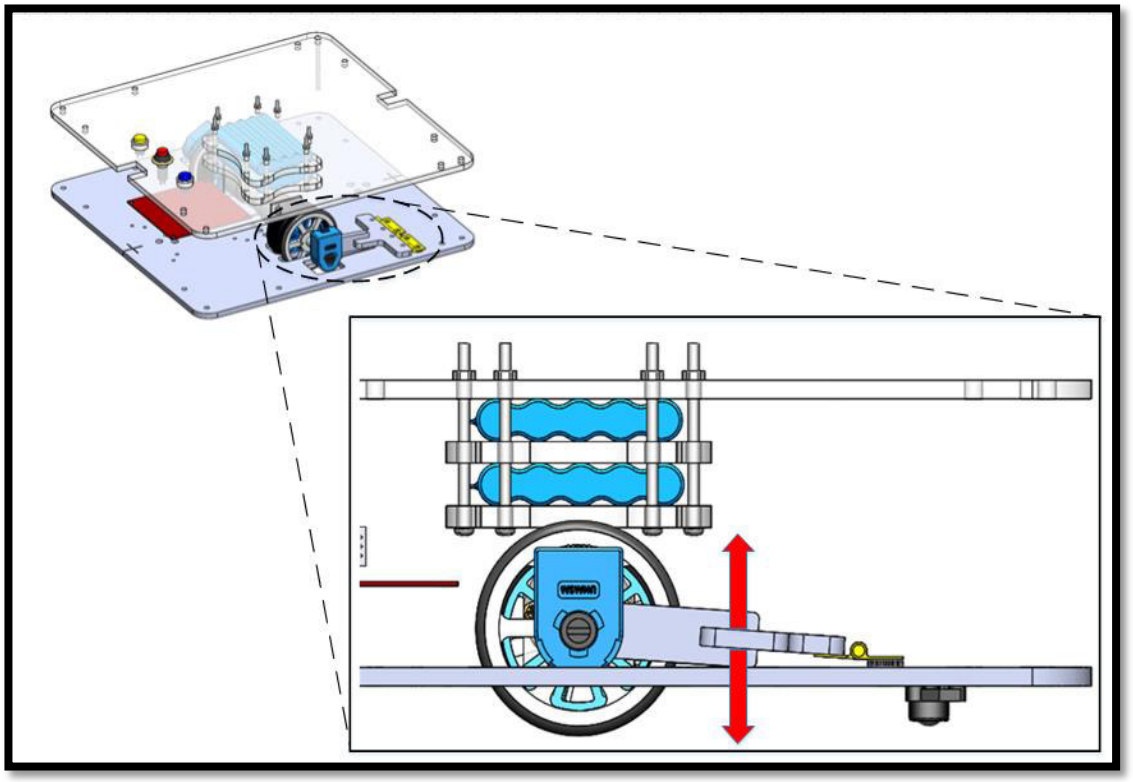


Figura 1.12.- Movimiento del mecanismo de la rueda del encoder.

Una vez sensado el desplazamiento del robot a través del movimiento angular de las ruedas de los encoders se realizará el análisis odométrico, el cual parte definiendo el movimiento genérico del robot, tal como se muestra en la figura 1.13. Donde el robot se desplaza de un punto a otro en un instante de tiempo (k), variando su ángulo de inclinación en Δθ. Las ruedas acopladas al encoder de la izquierda y derecha se desplazarán una distancia L1 y L2 respectivamente. Estas distancias serán detectadas por los encoders y generarán una cantidad de ticks las cuales llamaremos T1 y T2.

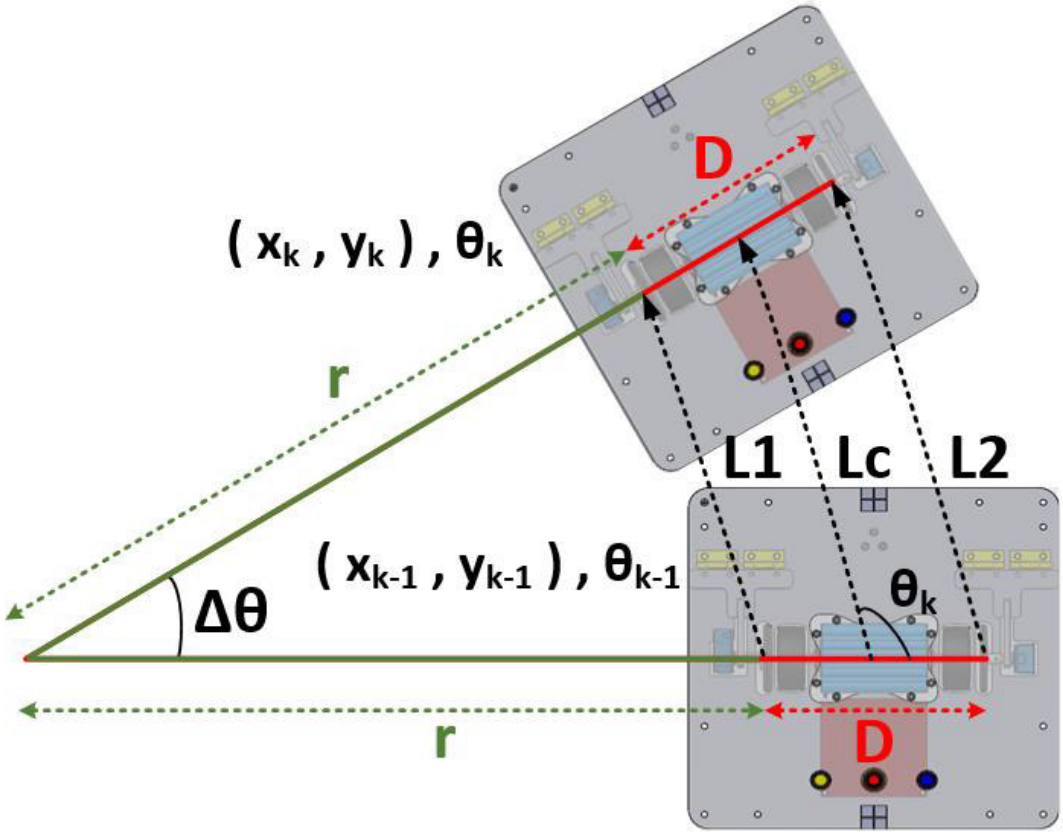


Figura 1.13.- Desplazamiento genérico del robot.

Empezaremos calculando las distancias recorridas por las ruedas de ambos encoders (L1 y L2), se utilizará la ecuación 1.3 desarrollada en el Capítulo 1, 2.Encoder.

$$L1 = \frac{2\pi \times R_W}{T_R} \times T_1 \ldots (1.4)$$

$$L2 = \frac{2\pi \times R_W}{T_R} \times T_2 \ldots (1.5)$$

Donde:

L1 = Distancia lineal recorrida por la rueda izquierda (m).

L2 = Distancia lineal recorrida por la rueda derecha (m).

T1 = Cantidad de ticks generados por el encoder izquierdo.

T2 = Cantidad de ticks generados por el encoder derecho.

$\mathbf{T_R}$ = Cantidad de ticks en una revolución.

Rw = Radio de la rueda (m).

De la figura 1.13 se deduce la siguiente relación entre las distancias recorridas, las cuales son arcos de circunferencia y el ángulo Δθ.

$$\frac{\boldsymbol{L1}}{\boldsymbol{r}} = \boldsymbol{\Delta\theta} \ldots (\mathbf{1.6})$$

$$\frac{\boldsymbol{L2}}{\boldsymbol{r}+\boldsymbol{D}} = \boldsymbol{\Delta\theta} \ldots (\mathbf{1.7})$$

Donde:

L1 = Distancia lineal recorrida por la rueda izquierda (m).

L2 = Distancia lineal recorrida por la rueda derecha (m).

Δθ = Ángulo de inclinación (rad).

r = Distancia del extremo del robot a la proyección (m).

D = Distancia entre las ruedas acopladas a los encoders (m).

Igualando las ecuaciones 1.6 y 1.7 , y despejando r se obtiene lo siguiente:

$$\boldsymbol{r} = \frac{\boldsymbol{D} \times \boldsymbol{L1}}{\boldsymbol{L2} - \boldsymbol{L1}} \ldots (\mathbf{1.8})$$

Reemplazando la ecuación 1.8 en 1.6 y despejando Δθ.

$$\boldsymbol{\Delta\theta} = \frac{\boldsymbol{L2} - \boldsymbol{L1}}{\boldsymbol{D}} \ldots (\mathbf{1.9})$$

El objetivo de este análisis es encontrar el desplazamiento y la orientación desarrollada por el centro del robot, el cual se encuentra en el punto medio entre las ruedas de los encoders. Para hallar el desplazamiento lineal del centro del robot (Lc) se empleará el mismo principio de la ecuación 1.6.

$$\frac{\boldsymbol{Lc}}{\boldsymbol{r}+\frac{\boldsymbol{D}}{\boldsymbol{2}}} = \boldsymbol{\Delta\theta} \ldots (\mathbf{1.10})$$

Despejando Lc en la ecuación 1.10 y reemplazando r y θ con las ecuaciones 1.8 y 1.9 respectivamente.

$$\boldsymbol{Lc} = \frac{\boldsymbol{L1}+\boldsymbol{L2}}{\boldsymbol{2}} \ldots (\mathbf{1.11})$$

En la ecuación 1.11 se expresa la relación entre la distancia recorrida por el centro del robot y las distancias recorridas por las ruedas de los encoders. Para encontrar las coordenadas (x,y) asociadas al desplazamiento se hará una descomposición vectorial a Lc.

$$\boldsymbol{Lcx} = \frac{\boldsymbol{L1}+\boldsymbol{L2}}{\boldsymbol{2}}\boldsymbol{cos}(\boldsymbol{\theta_k}) \ldots (\mathbf{1.12})$$

$$\boldsymbol{Lcy} = \frac{\boldsymbol{L1}+\boldsymbol{L2}}{\boldsymbol{2}}\boldsymbol{sen}(\boldsymbol{\theta_k}) \ldots (\mathbf{1.13})$$

Reemplazando las ecuaciones 1.4 y 1.5 en 1.9 , 1.12 y 1.13 se obtiene la variación angular (Δθ), y las variaciones en las coordenadas en función de los tikcs generados por los dos encoders.

$$\Delta\boldsymbol{\theta} = \frac{\boldsymbol{2\pi}\times\boldsymbol{R_W}}{\boldsymbol{T_R}\times\boldsymbol{D}}\times(\boldsymbol{T_1}-\boldsymbol{T_2}) \ldots (\mathbf{1.14})$$

$$\Delta\boldsymbol{x} = \frac{\boldsymbol{\pi}\times\boldsymbol{R_W}}{\boldsymbol{T_R}}\times(\boldsymbol{T_1}+\boldsymbol{T_2})\times\mathbf{cos}(\boldsymbol{\theta_k}) \ldots (\mathbf{1.15})$$

$$\Delta\boldsymbol{y} = \frac{\boldsymbol{\pi}\times\boldsymbol{R_W}}{\boldsymbol{T_R}}\times(\boldsymbol{T_1}+\boldsymbol{T_2})\times\mathbf{sen}(\boldsymbol{\theta_k}) \ldots (\mathbf{1.16})$$

Conociendo las variaciones en la orientación y las coordenadas (ecuaciones 1.14, 1.15 y 1.16) se podrán generalizar a través de las siguientes ecuaciones en diferencias:

$$\boldsymbol{\theta_k = \theta_{k-1} + \Delta\theta} \ldots (\mathbf{1.17})$$

$$\boldsymbol{x_k = x_{k-1} + \Delta x} \ldots (\mathbf{1.18})$$

$$\boldsymbol{y_k = y_{k-1} + \Delta y} \ldots (\mathbf{1.19})$$

Las ecuaciones 1.17, 1.18 y 1.19 se emplearán el algoritmos de iteración para calcular los parámetros odométricos en forma discreta.

4. Controlador PID

4.1. Introducción

Las estrategias de control PID se incorporaron en el ambiente industrial en el primer cuarto de este siglo, con un esquema de ajuste puramente empírico [8]. En la actualidad, y pese al sorprendente desarrollo de la teoría de control y del soporte tecnológico necesario para su implementación, el controlador de estructura PID se emplea casi con exclusividad en el ambiente industrial de todo el mundo, en particular para controlar procesos térmicos y químicos [9].

A lo largo de varias décadas, el controlador PID ha demostrado un comportamiento robusto. En su versión más simple, el controlador PID posee solo tres parámetros de ajuste y por consiguiente su comportamiento depende éstas, de la complejidad del proceso a controlar y de las perturbaciones a rechazar.

El PID es un controlador (C(s)) que recibe como señal de entrada al error (e(t)), el cual es la diferencia entre el setpoint o referencia (r(t)) y la realimentación de la señal de salida de la planta o proceso (y (t)). El resultado que genera el controlador PID toma el nombre de u(t) o señal de control, esta señal ingresará a la planta o proceso (G) tal como se muestra en la figura 1.14. El objetivo del controlador es tratar que la señal de error e(t) sea igual a cero, quiere decir, que la salida del proceso y(t) sea igual a la referencia r(t).

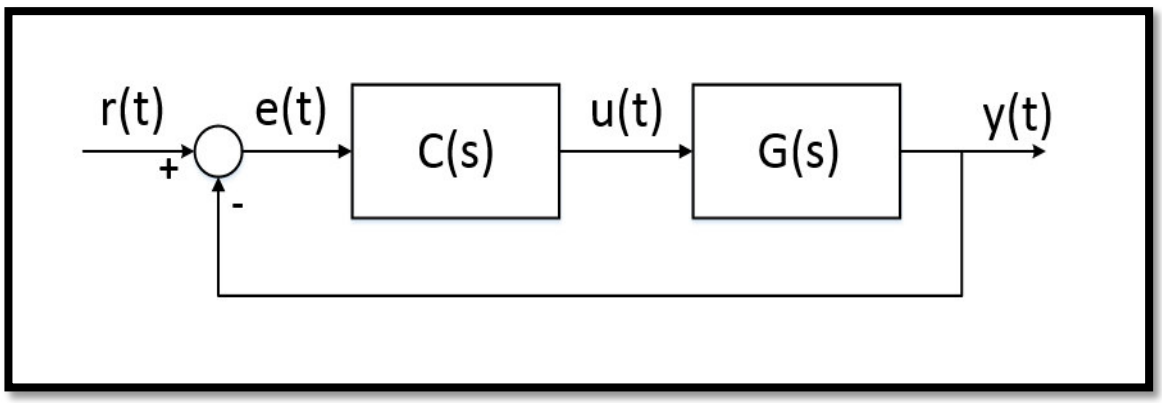


Figura 1.14.- Esquema básico de un sistema de control.

La estructura clásica de un PID, la cual se muestra en la figura 1.15, consta de tres acciones de control: Acción proporcional (Kp), acción integral (Ki.s) y acción derivativa (Kd/s). La suma de estas acciones genera la señal de control u(t), la cual se expresa matemáticamente en la ecuación 1.20.

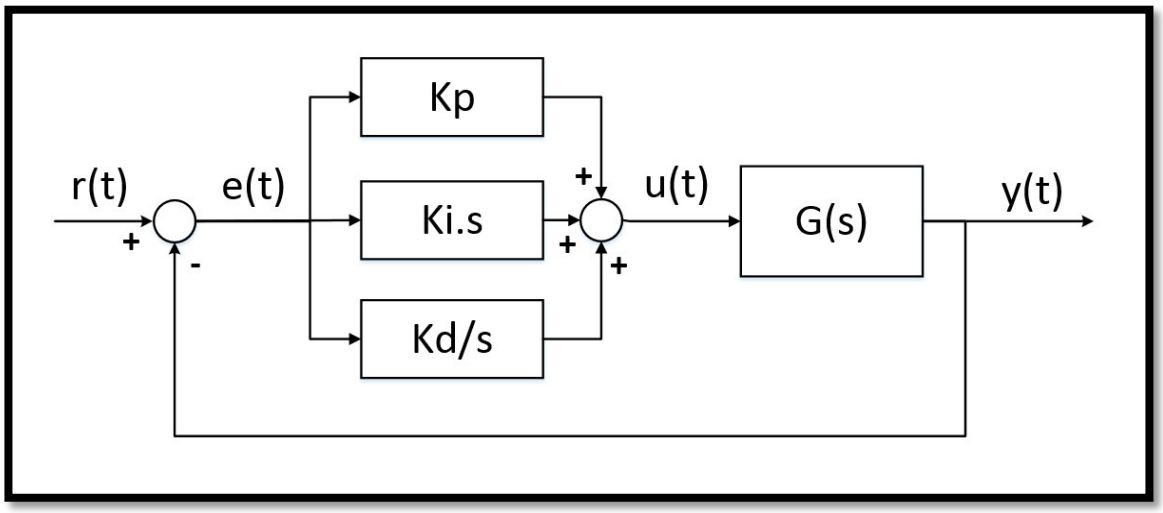


Figura 1.15.- Estructura clásica de un PID.

$$u(t) = Kp \times e(t) + Ki \times \int_0^t e(t)dt \; + \; Kd \times \frac{de(t)}{dt} \quad \ldots \; (1.20)$$

La sintonía del controlador se reduce al ajuste de la constante proporcional (Kp), integral (Ki) y derivativa (Kd). Una correcta selección de estos parámetros generará que el lazo de control corrija la salida del proceso eficazmente y en el menor tiempo posible [10]. Esto evitará que la salida del sistema oscile o sea inestable.

Acción Proporcional

La respuesta proporcional es la base de los tres modos de control, si los otros dos, control integral y control derivativo están presentes, éstos son sumados a la respuesta proporcional. Proporcional significa que el cambio presente en la salida del controlador es algún múltiplo del error e(t). Este múltiplo es llamada “constante proporcional” del controlador (Kp). La función matemática de esta acción se muestra en la ecuación 1.21.

$$p(t) = Kp \; \times \; e(t) \; \ldots (1.21)$$

Las características en el sistema cuando la constante proporcional Kp aumenta son las siguientes:

- El error en estado estacionario disminuye.
- El proceso responde más rápidamente.
- La sobre oscilación y las oscilaciones aumentan.

Acción Integral

La acción integral genera una respuesta en función a la integral del error e(t). Esta acción tiene como propósito disminuir o eliminar el error en estado estacionario, provocado por la acción proporcional. El control integral actúa cuando hay una desviación entre la salida del proceso y(t) y la referencia r(t), integrando esta desviación en el tiempo y sumándola a la acción proporcional. El error es integrado, lo cual tiene como objetivo promediarlo o sumarlo por un período de tiempo determinado, luego es multiplicado por la "constante integral" (Ki) tal como se muestra en la ecuación 1.22. Posteriormente, la respuesta integral es adicionada a la acción proporcional con el propósito de obtener una respuesta estable del sistema sin error estacionario.

$$i(t) = Ki \times \int_0^t e(t)dt \quad \dots \ (1.22)$$

Acción Derivativa

La acción derivativa genera una respuesta en función a la derivada del error e(t) o velocidad de cambio del error. Esta acción se manifiesta cuando hay una variación en el valor absoluto del error. Si el error es constante su aporte será nulo. El objetivo es mantener el error al mínimo corrigiéndolo proporcionalmente a la velocidad que se produce, de esta manera se evita que el error se incremente. En la ecuación 1.23 se muestra la representación matemática de esta acción, en la cual se multiplica la derivada del error con la "constante derivativa" (Kd).

$$d(t) = Kd \times \frac{de(t)}{dt} \quad \dots \ (1.23)$$

4.2. PID discreto

En la figura 1.16 se muestra el diagrama básico de un sistema de control discreto, cuyo objetivo es lograr que la salida del proceso Y(k) sea igual o muy cercana a la señal de referencia R(k). Para realizar esta tarea se emplea un controlador discreto C, el cual emplea un algoritmo de control implementado en un dispositivo digital como una computadora, microcontrolador o algún otro. Este algoritmo se ejecuta cada periodo de tiempo T y genera una señal discreta U(k). Dependiendo del tipo de actuador la señal U(k) puede ser convertida a una señal analógica o modulada en ancho de pulso (PWM) [11].

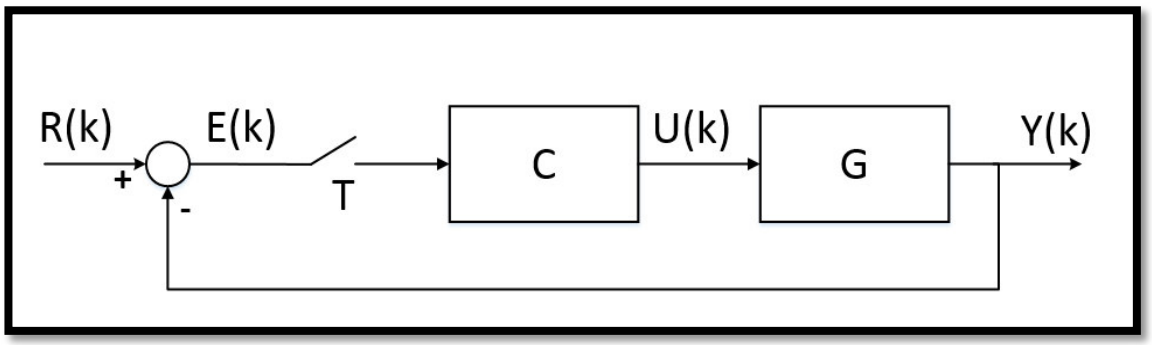


Figura 1.16. Diagrama básico de control discreto.

El diseño del controlador digital suele ser realizado empleando técnicas propias de los sistemas muestreados. Potencialmente, estas técnicas permiten obtener controles más versátiles que los que se pueden conseguir con compensadores analógicos. Sin embargo, existen aplicaciones donde los controladores analógicos han demostrado trabajar satisfactoriamente, razón por la cual en muchas de estas aplicaciones se prefiere diseñar los controladores digitales directamente como una aproximación de los controladores analógicos [12]. Este es, por ejemplo, el caso del controlador PID cuya implementación digital es solo una aproximación numérica de su ecuación integro-diferencial (ecuación 1.20).

Acción Proporcional discreta

El índice "k" es la representación digital del tiempo discreto, el cual está asociado con el tiempo continuo "t" a través de la siguiente relación:

$$t = k \times T$$

Donde T es el periodo de muestreo. La componente proporcional del controlador PID en tiempo continuo dada por la ecuación 1.21 se representada en tiempo discreto en la ecuación 1.24.

$$P(k) = Kp \times e(k) \ \ldots (1.24)$$

Acción Integral discreta

La integral se calcula como el área bajo la curva. Para hallar esta área existen diversos métodos, uno de ellos es la aproximación trapezoidal [13]. Esta consiste en calcular el área tomando las señales de error E(k) y E(k-1) como la base mayor y menor de un trapecio; y el periodo T como la altura, tal como se muestra en la figura 1.17.

El área del trapecio de muestra en la ecuación 1.125

$$S = \frac{B + b}{2} \times h \ \ldots (1.25)$$

Donde:

S = Área del trapecio.

B = Base mayor.

b = Base menor.

h = Altura.

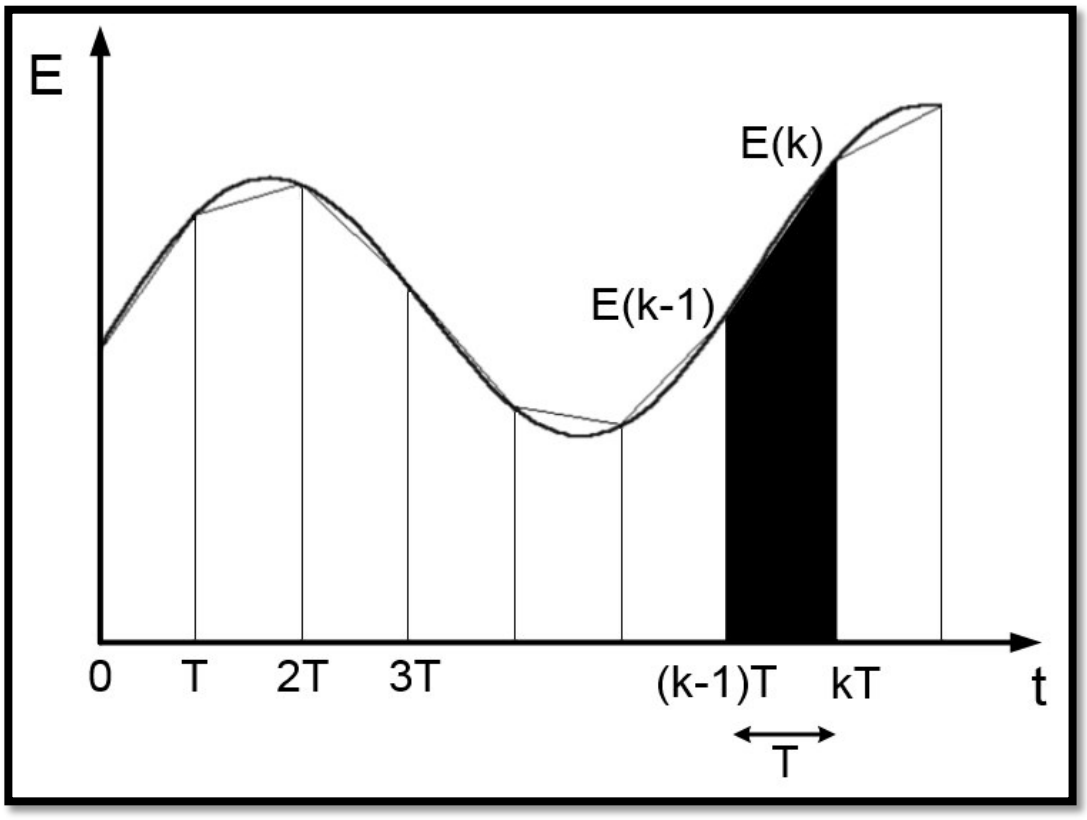


Figura 1.17. Aproximación de la acción integral.

Si reemplazamos las variables de la ecuación 1.25 con los valores mostrados en la figura 1.17, se obtendrá la ecuación 1.26:

$$S = \frac{E(k) + E(k-1)}{2} \times T \quad \ldots (1.26)$$

En la ecuación 1.26 se obtuvo el área de la curva en un instante de tiempo T, para encontrar el área total se empleará una sumatoria de todas las áreas en cada instante de tiempo. El resultado de esta sumatoria multiplicado por la constante integral Ki (ecuación 1.27), será la aproximación discreta de la acción integral (ecuación 1.22).

$$I(k) \cong Ki \sum_{i=0}^{k} \frac{T}{2} \times (E_i + E_{(i-1)}) \quad \ldots (1.27)$$

La ecuación 1.27 no es útil para una aplicación en tiempo real debido a que requiere que se almacenen todos los datos muestreados para poder ejecutar la sumatoria. Por este motivo se empleará un algoritmo recursivo en la cual solo se requiera almacenar dos datos en la memoria del dispositivo digital en el que se implemente el PID discreto.

Se quitará un término de la sumatoria de la ecuación 1.27.

$$I(k) \cong Ki \sum_{i=0}^{k-1} \frac{T}{2} \times (E_i + E_{(i-1)}) + Ki \frac{T}{2} \times (E_k + E_{(k-1)}) \quad \ldots (1.28)$$

Finalmente se obtiene la ecuación 1.29 en formato recursivo, la ventaja de esta nueva aproximación es que solo se requiere almacenar el valor del error y acción integral del periodo T anterior.

$$I(k) \cong I(k-1) + Ki \frac{T}{2} \times (E_k + E_{(k-1)}) \quad \ldots (1.29)$$

Acción Derivativa discreta

La discretización de la acción derivativa en tiempo continuo de la ecuación 1.23 se muestra en la ecuación 1.30. Esta aproximación (ver figura 1.18) se calcula como la diferencia entre el error actual y el error anterior entre la diferencia de tiempo (tiempo de muestreo T).

$$D(k) \cong Kd \frac{(E_k - E_{(k-1)})}{T} \quad \ldots (1.30)$$

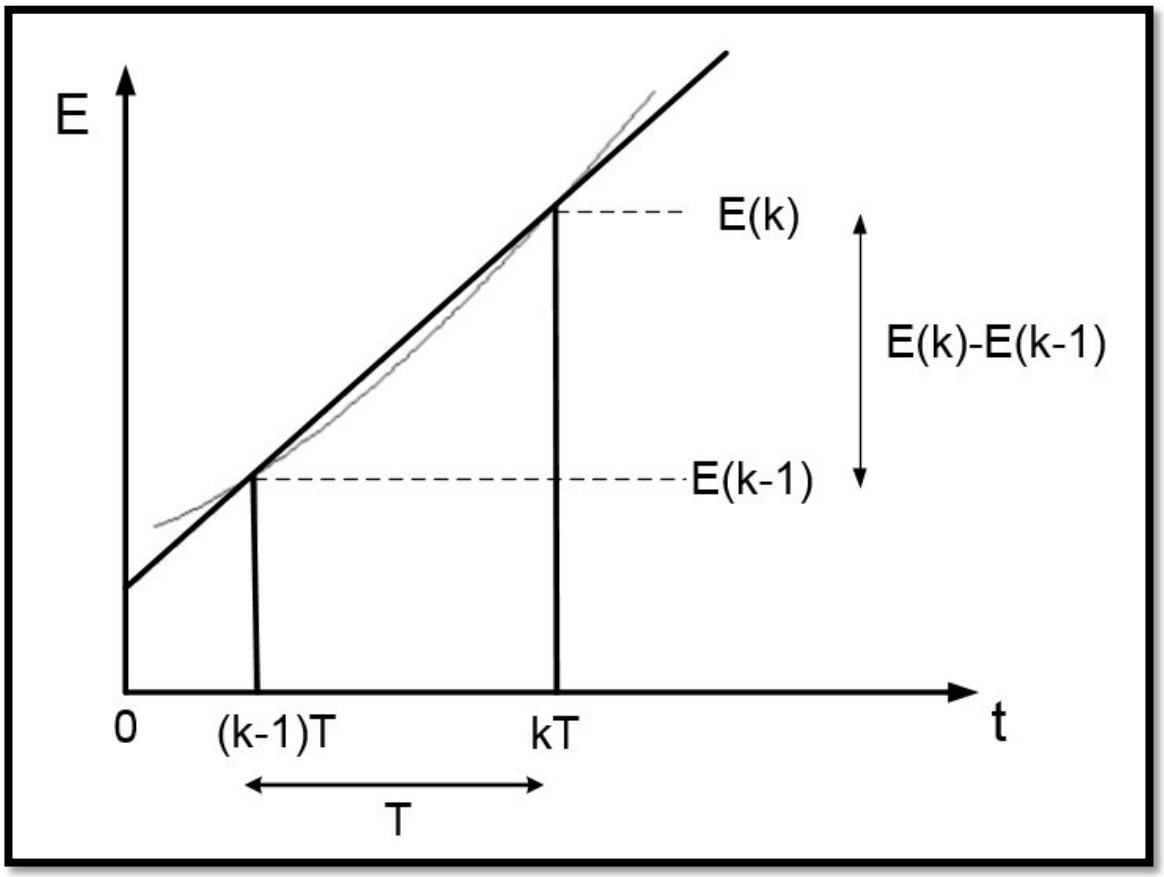


Figura 1.18. Aproximación de la acción derivativa.

Finalmente la discretización de la señal de salida de control u(t) se compone por la suma de las tres acciones de control (proporcional, integral y derivativa) tal como se muestra en la ecuación 1.31.

$$U(k) = P(k) + I(k) + D(k) \ldots (1.31)$$

Reemplazando las ecuaciones 1.25, 1.29 y 1.30 en la ecuación 1.31 se obtiene la señal de control discreta U(k) en la ecuación 1.32.

$$\boldsymbol{U(k) = Kp \times e(k) + I(k-1) + Ki\frac{T}{2} \times (E_k + E_{(k-1)}) + Kd\ \frac{(E_k - E_{(k-1)})}{T} \ldots (1.32)}$$

Capítulo II
Descripción del Diseño del Robot

1. Estructura mecánica

La primera etapa para la construcción del robot es el diseño de la estructura mecánica. Sobre dicha estructura se ensamblarán todos los elementos (ver figura 2.1) tanto de control (tarjeta electrónica), adquisición (encoders) y actuadores (motores). Razón por la cual se debe tener una vista previa del diseño final para tener la certeza de que todos los elementos encajaran en sus posiciones. Por tal motivo se utilizó el software SolidWorks para el diseño y simulación de las piezas de la estructura mecánica.

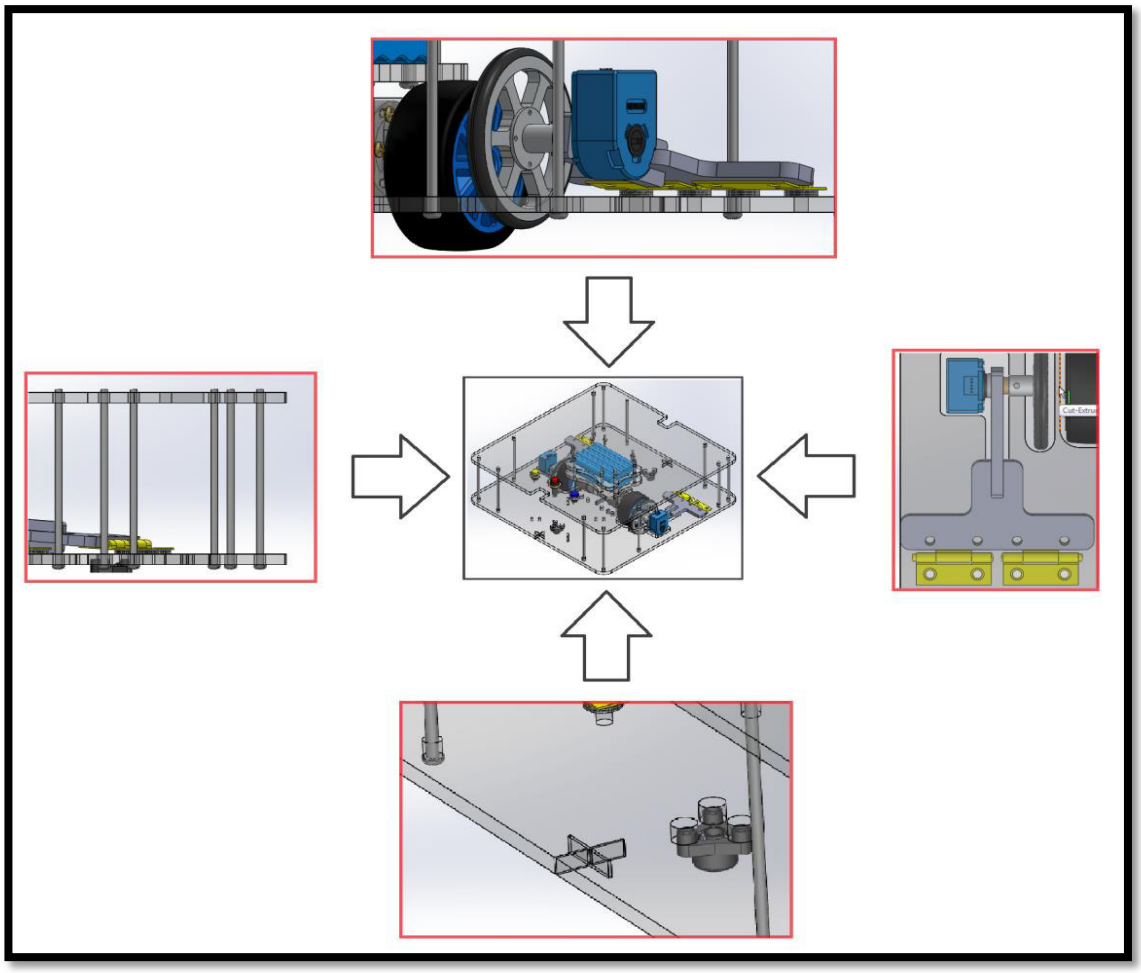


Figura 2.1.- Elementos de la estructura del robot.

El robot móvil posee las siguientes características, las cuales se ven reflejadas en las vistas de las figuras 2.2, 2.3, 2.4, 2.5 y 2.6.

- El chasis fue construido en acrílico de 6mm de espesor mediante corte laser.
- Posee un tamaño de 36x36x11 cm.
- Sistema independiente para los encoders.
- Soporte para las baterías en la parte superior.
- Dos marcas en forma de cruz para realizar las calibraciones.
- Dos ruedas de libre giro colocados en los extremos del robot para mayor estabilidad.

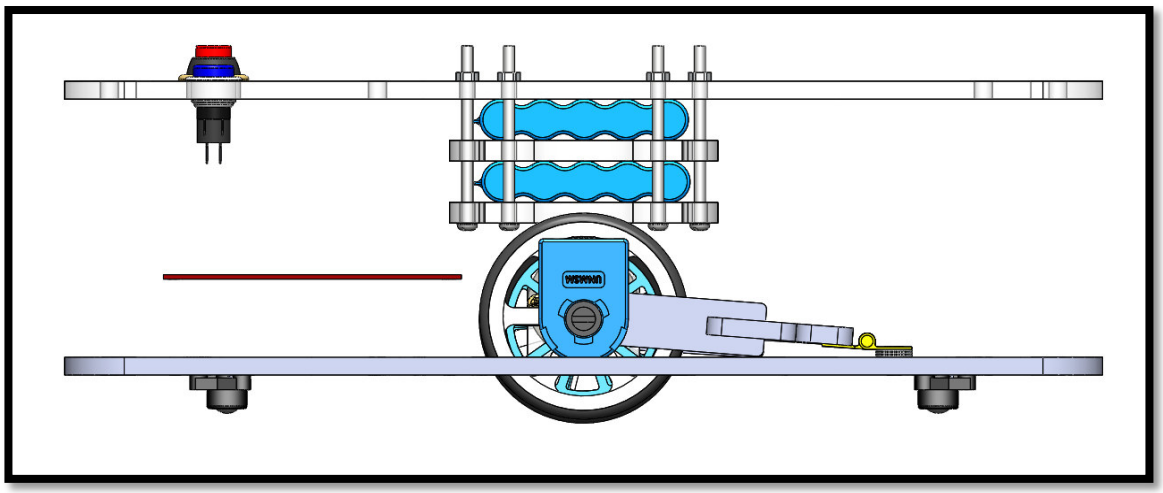


Figura 2.2.- Vista lateral del robot.

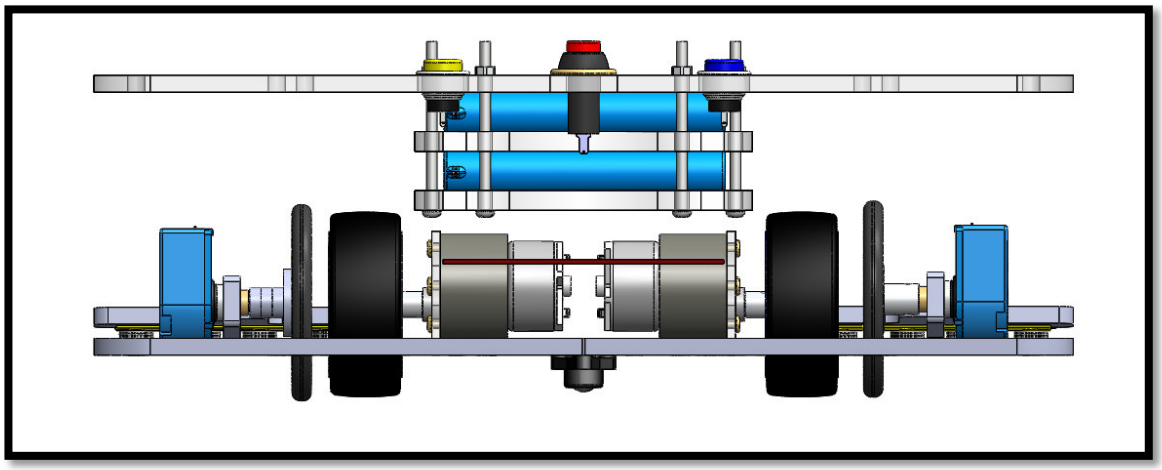


Figura 2.3.- Vista frontal del robot.

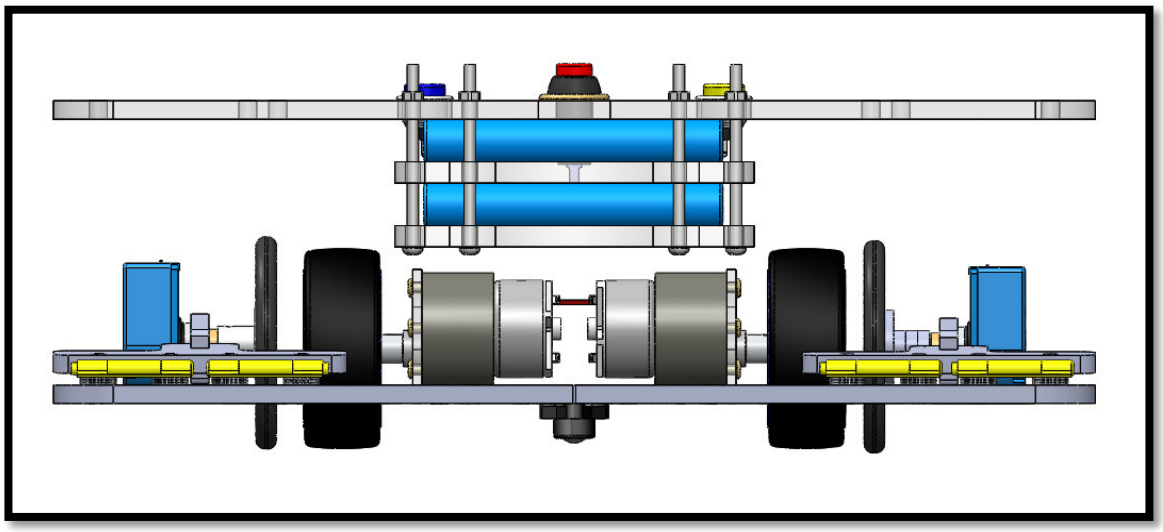


Figura 2.4.- Vista posterior del robot.

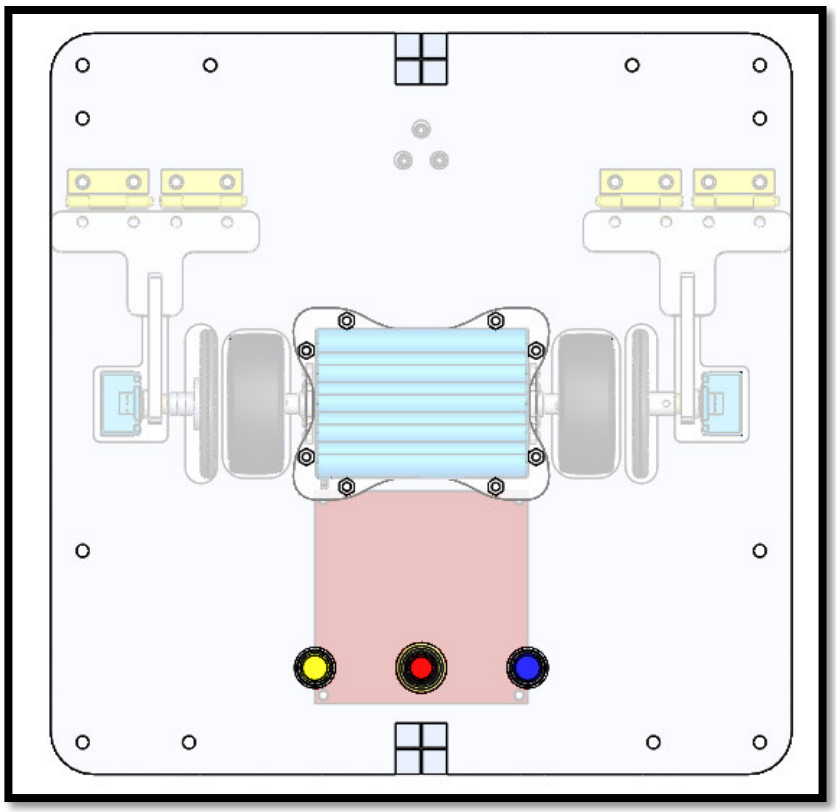


Figura 2.5.- Vista superior del robot.

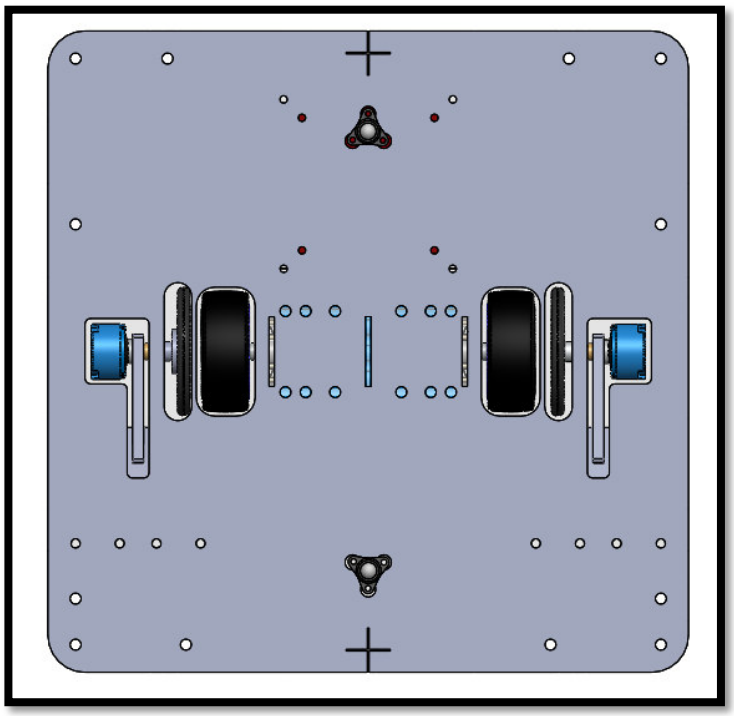


Figura 2.6.- Vista inferior del robot.

Para evitar falsas lecturas de los encoders generadas por el patinaje de las ruedas debido a una sobre aceleración de los motores, suelos resbaladizos o por la acción de la inercia se implementó un mecanismo en el cual los encoders están acoplados a ruedas independientes de los motores, además poseen un movimiento libre en el vertical que les permite un contacto permanente con la superficie (ver figura 2.7), tal como se detalló en el capítulo 1, 3.Odometría.

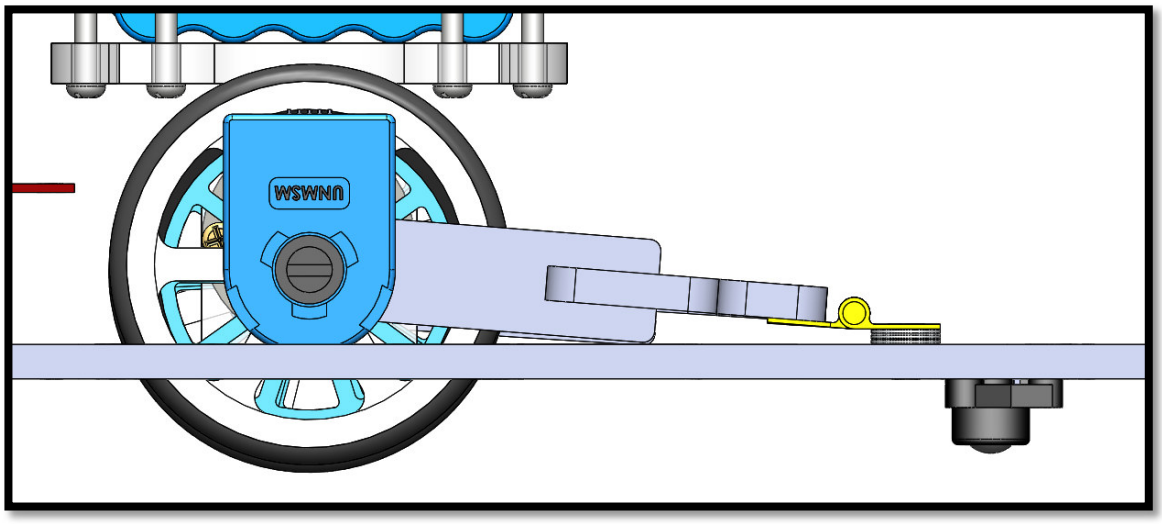


Figura 2.7.- Vista 1 del mecanismo del encoder independiente.

El encoder está acoplado a una rueda que girará de acuerdo al movimiento del robot. Este acople esta sujetado por un brazo que se une con la base de la estructura a través de un sistema de bisagras (figura 2.8 y 2.9).

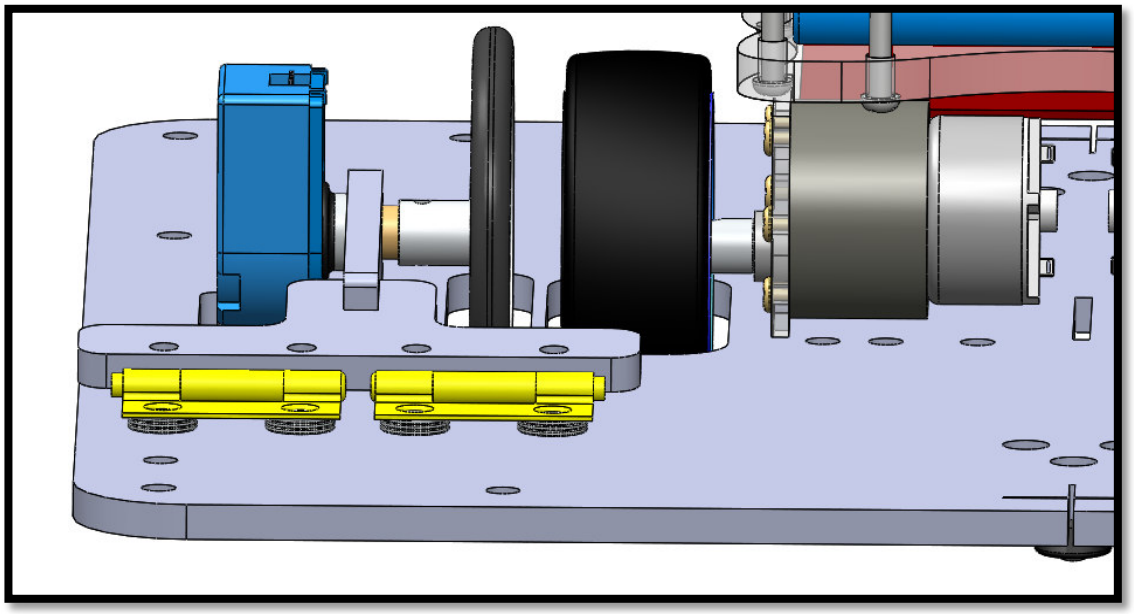


Figura 2.8.- Vista 2 del mecanismo del encoder independiente.

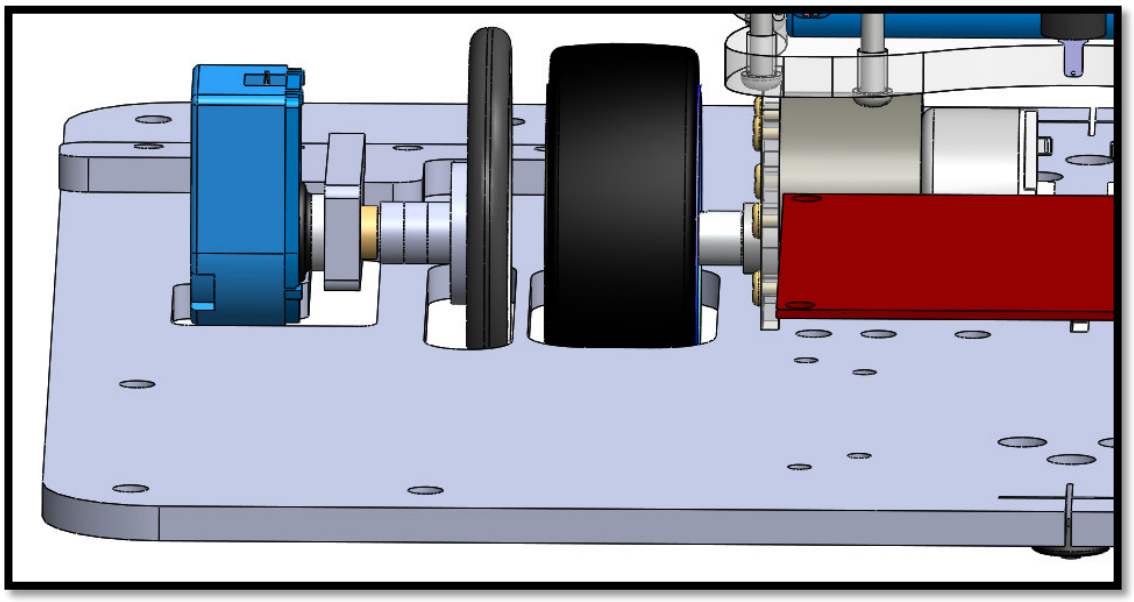


Figura 2.9.- Vista 3 del mecanismo del encoder independiente.

Adicionalmente la estructura del robot cuenta con dos marcas en forma de cruz (ver figura 2.10) para facilitar las calibraciones y mediciones.

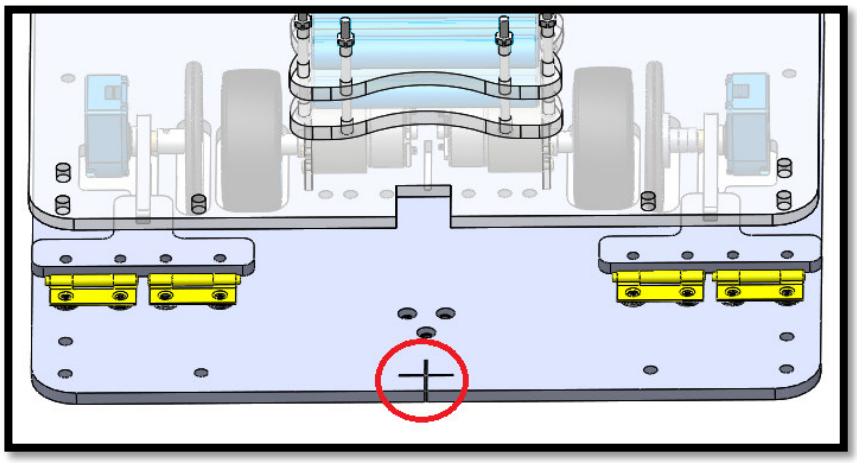


Figura 2.10.- Marcas en forma de cruz.

Finalmente se implementó la estructura del robot y se ensamblaron todos los componentes en función al diseño previo en SolidWorks. El robot implementado se puede observar en la figura 2.11.

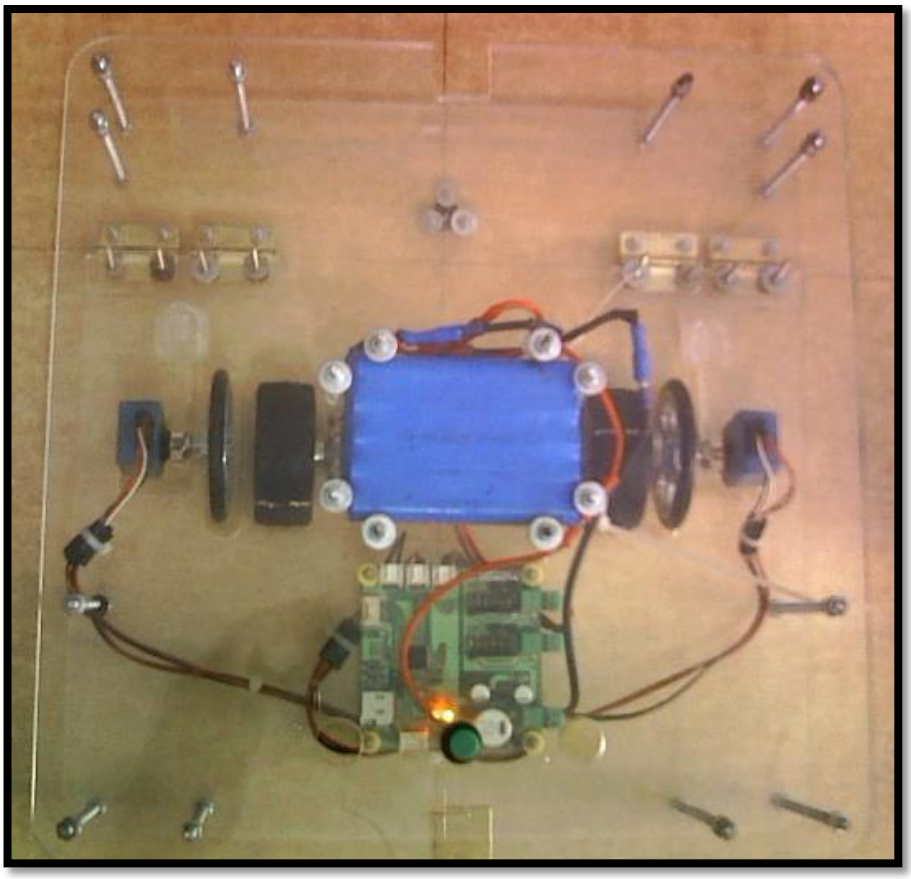


Figura 2.11. -Estructura del robot implementada.

2. Tarjeta electrónica

El robot cuenta con una tarjeta electrónica que controla y administra todo el sistema. De acuerdo a la figura 2.12, la tarjeta presenta tres etapas: alimentación, lógica y de potencia. Cada una de estas etapas está dividida y cumple con una función específica. Además la tarjeta cuenta con un sistema de comunicación serial para establecer una conexión con una PC.

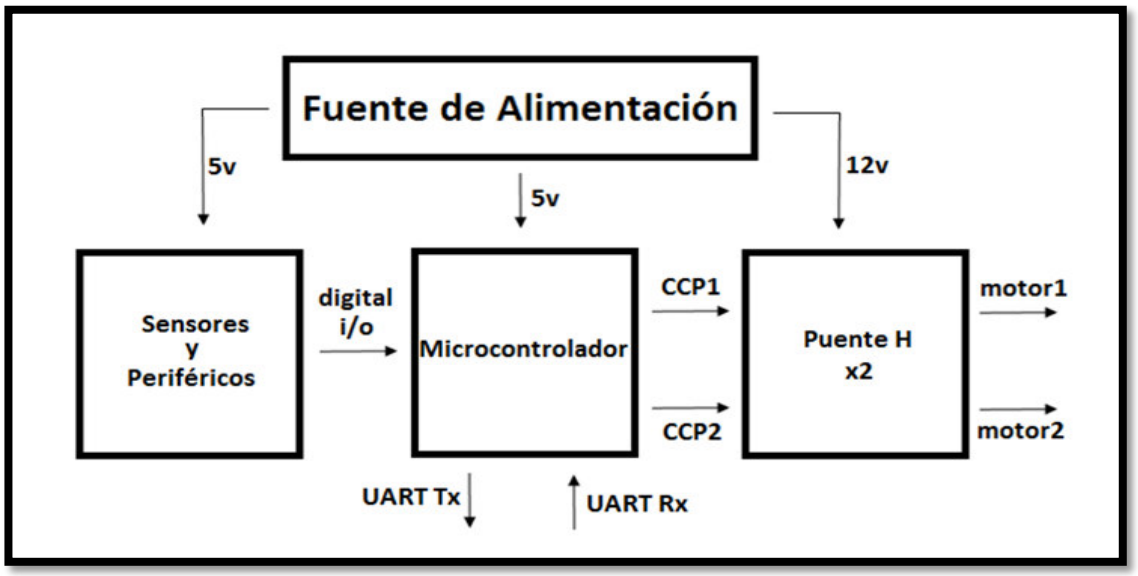


Figura 2.12.- Etapas de la tarjeta electrónica.

2.1 Etapa de alimentación

El robot requiere dos niveles de tensión para su funcionamiento, ya que cuenta con componentes digitales (microcontrolador, encoders, leds, conversor serial-USB) que se alimentan con 5 voltios y con actuadores (motores) que operan a 12 voltios como se mostró en la figura anterior. La etapa de alimentación se encarga de regular la tensión de 12V a 5V.

La energía es suministrada por baterías recargables NI-MH (ver figura 2.13). Estas baterías, utilizan un ánodo de oxidróxido de níquel (NiOOH), lo que hace que sean más amigables para el medio ambiente y de gran capacidad de carga. Su autonomía es de 1.3 Ah, por lo que se utilizaron dos baterías en paralelo para duplicar la duración.

Figura 2.13.- Batería NI-MH.

Para realizar la regulación se utilizó el C.I. L5973D (ver Anexo A.2, L5973D) el cual convertirá los 12V del arreglo en paralelo de las baterías a 5V (ver figura 2.14). Se utilizó este regulador debido a su alto rendimiento (90%), mejorando la autonomía de funcionamiento y operación del robot. Además posee un amplio rango de voltaje de entrada, dando la opción de aumentar el nivel de tensión sin alterar su funcionamiento. Otra de las características que lo hacen ideal para esta aplicación es su reducido tamaño; así como su package para aplicaciones con tecnología SMD.

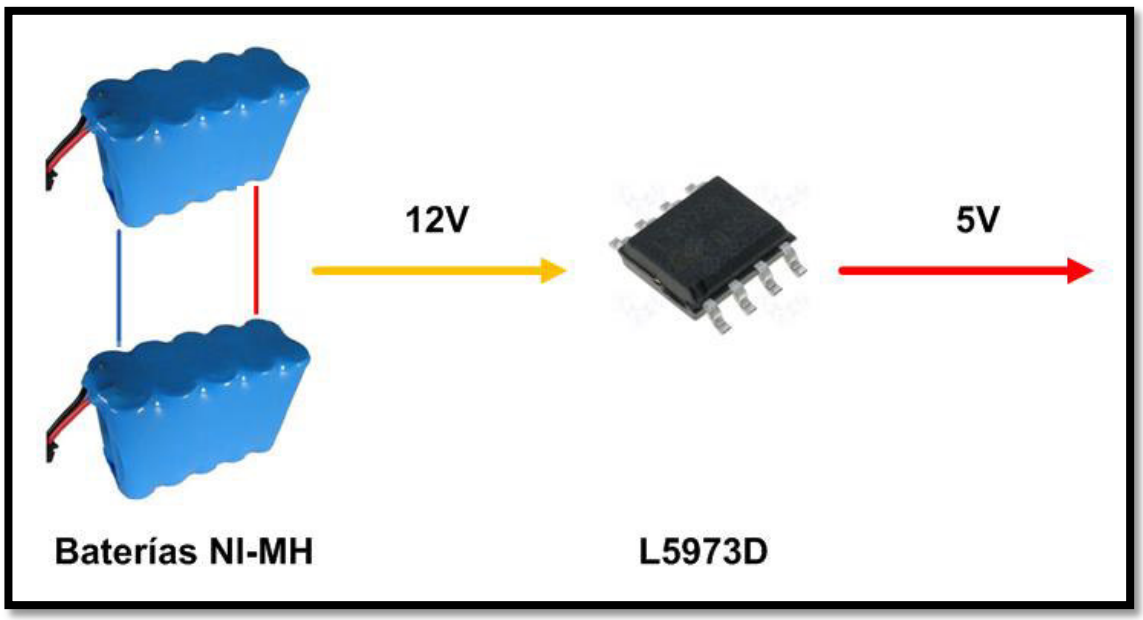


Figura 2.14.- Esquema de la etapa de alimentación.

2.2 Etapa lógica

La principal función de la tarjeta electrónica es la de controlar y administrar al robot. Para este propósito se empleó un dsPIC30F4011, como unidad de procesamiento y control de las señales. El dsPIC recibe los trenes de pulsos generados por los encoders y mediante una serie de algoritmos programados generará una señal de control que actuará sobre los motores, tal como se observa en la figura 2.15. Esto quiere decir que el control será embebido dentro del robot para incrementar la robustez del sistema. A diferencia que el control se haga remotamente (por ejemplo una PC), lo cual provocaría tiempos largos de respuesta y una dependencia con otro dispositivo.

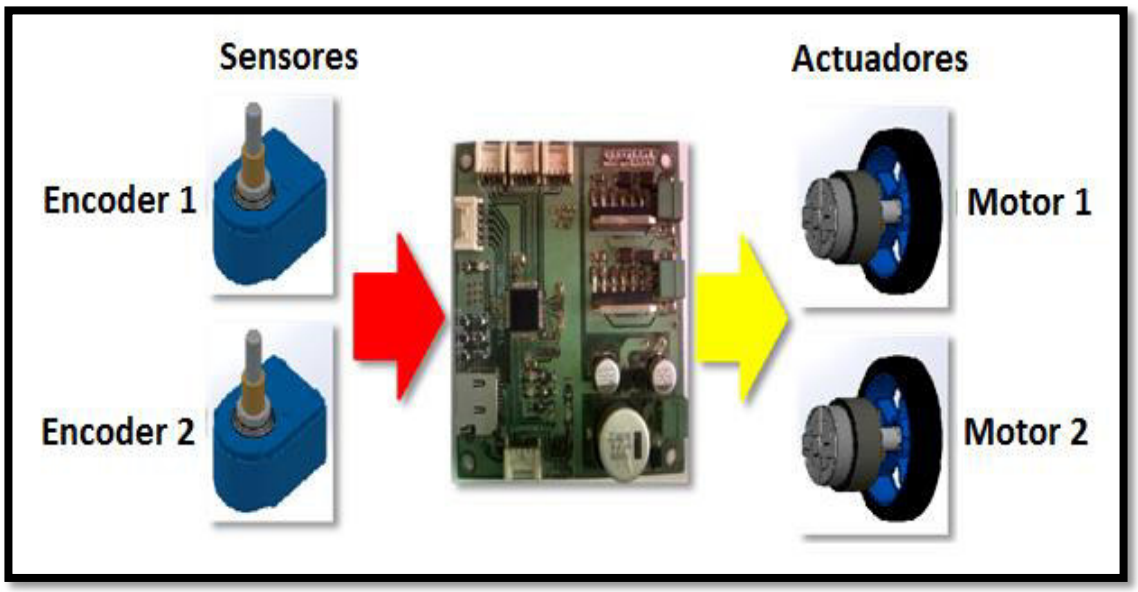


Figura 2.15.- Entrada y salidas lógicas del dsPIC.

El dsPIC usado para esta aplicación es el dsPIC30F4011, este microcontrolador de la marca Microchip pertenece a la familia DSC (ver Anexo A.1, dsPIC30F4011), los cuales son muy potentes para aplicaciones de control y procesamiento de datos, así como muy flexible con el uso de los periféricos. Dentro de esta familia existen dos categorías, los dsPIC30F y los dsPIC33F (ver figura 2.16). Se decidió utilizar los dsPIC30F debido a que su alimentación y periféricos trabajan a 5 voltios, al igual que los encoders, puente h y conversor serial-USB. Con el fin de uniformizar la tensión a 5 voltios se empleó dicha categoría.

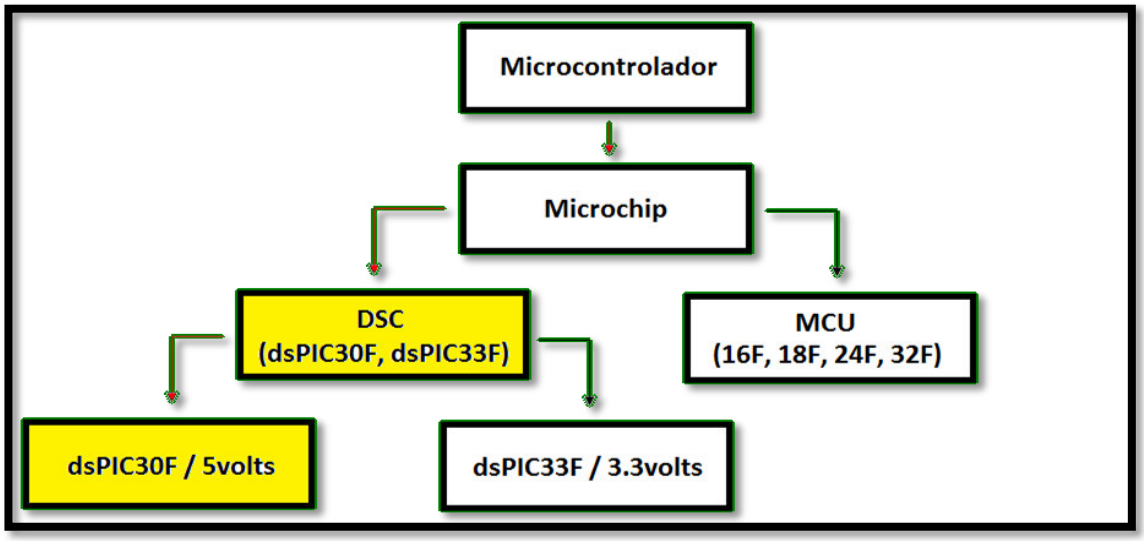


Figura 2.16.- Selección del microcontrolador.

El dsPIC cuenta con los periféricos que se muestran en la figura 2.17. Estos módulos facilitan la programación del microcontrolador ya que cuentan con arquitecturas prediseñadas de propósito específico como: puerto serial para realizar la comunicación con elementos externos (por ejemplo una PC); ICSP (in circuit serial programing) para realizar la grabado del programa en el dsPIC a través de 5 pines; entradas analógicas de 10 bits de resolución utilizados para la lectura del nivel de tensión de la batería; entradas y salidas digitales que se usarán para la lectura de los pulsos de los encoders y visualización de avisos a través de leds; CCP1 y CCP2 son los módulos que generan la señal de PWM (modulación por ancho de pulso) los cuales se emplearon para el control de velocidad de los motores.

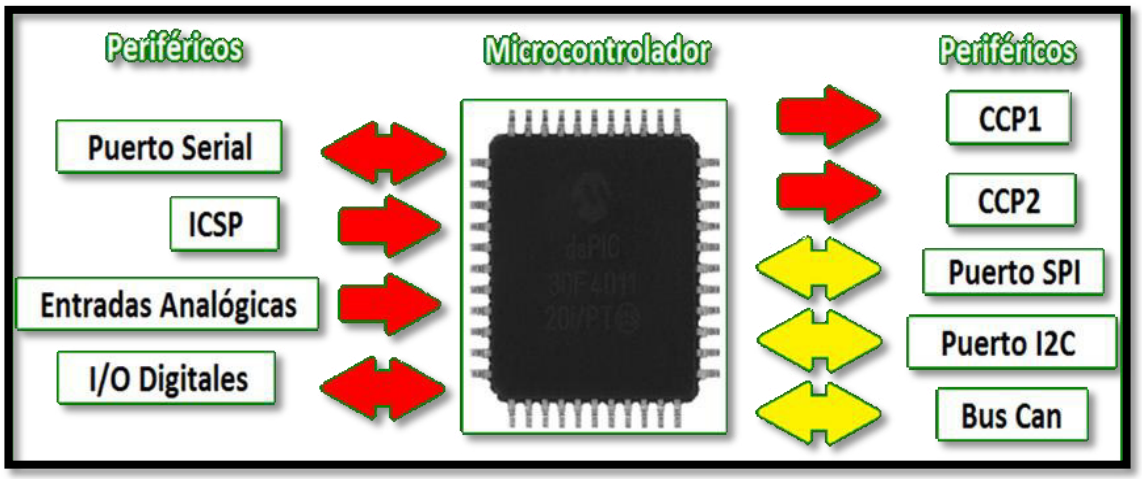


Figura 2.17.- Periféricos del microcontrolador.

2.3 Etapa de potencia

La etapa de potencia consiste en amplificar la corriente de la señal de control generada por el dsPIC para el accionamiento de los dos motores DC (ver figura 1.18) que generan el movimiento del robot. Dichos motores operan a 12vdc, 4A, 1.12Kg-cm y 300 RPM, por lo que requieren un driver que soporte estos parámetros. El driver usado es el L6203 (ver Anexo A.3, L6203), el cual funciona como interfaz (puente H) entre el dsPIC y los motores elevando la señal de control generada por el controlador (5vdc) a la tensión nominal del motor (12vdc).

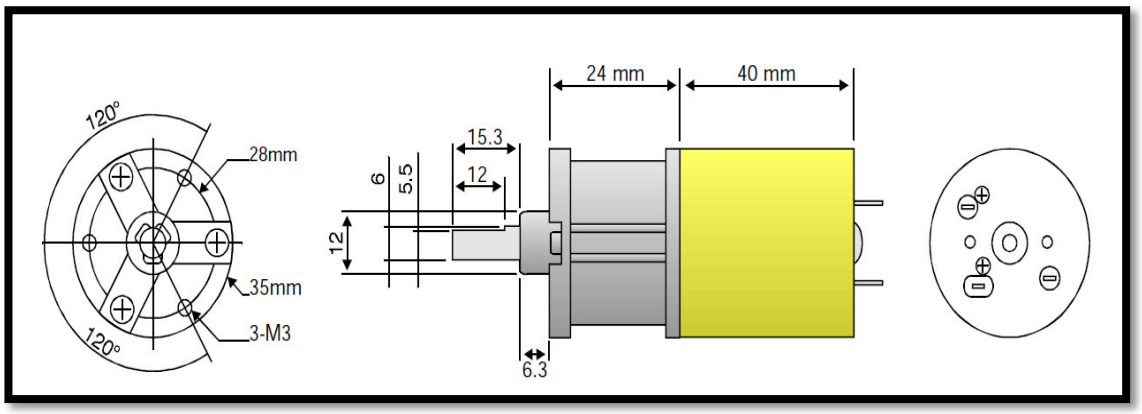


Figura 2.18.- Motor Lynxmotion.

Se utiliza un driver por cada motor para controlar su sentido de giro, así como su velocidad. El dsPIC emplea cuatro señales para realizar las conmutaciones de giro de ambos motores, tal como se observa en la figura 2.19. La lógica de control de los motores será explicado en los capítulos posteriores.

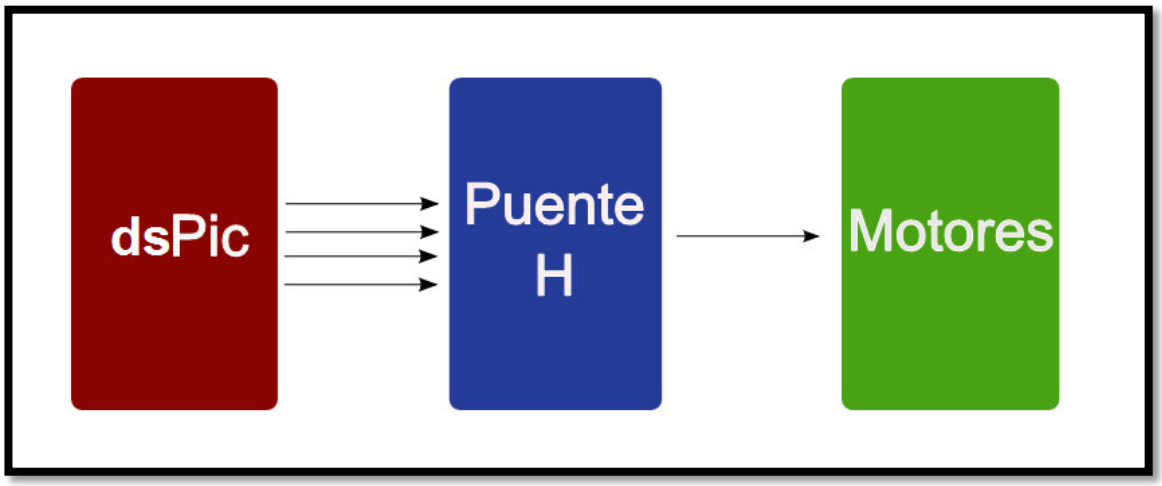


Figura 2.19.- Señales del dsPIC al L6203.

En el diseño de la tarjeta electrónica se emplearon todas las consideraciones ya explicadas previamente como el tamaño, distribución de componentes, sistema de alimentación, lógica y potencia. Además se utilizó la tecnología SMD (componentes superficiales) para tener dimensiones más pequeñas y optimizar el espacio de tarjeta. La talla utilizada para los componentes pasivos (resistencias, condensadores, bobinas y leds) es la 1206. Además se empleó el software Eagle para el diseño del layout en doble capa (ver figura 2.20).

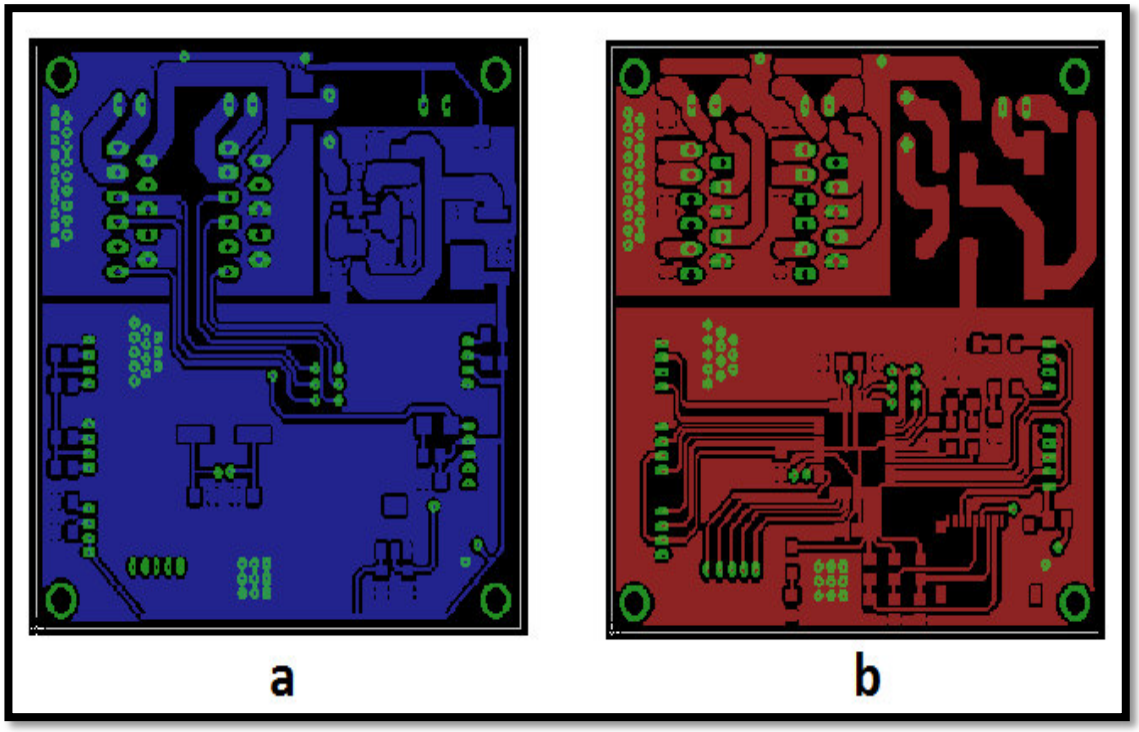


Figura 2.20.- Placa en doble capa; a: cara inferior, b: cara superior

Las dimensiones de la tarjeta electrónica son de 8x8 cm. Además cuenta con protección de alimentación inversa para evitar la incorrecta polarización al momento de conectar la batería. También posee con indicadores (leds) para mostrar diferentes estados durante la configuración y operación del robot.

La tarjeta cuenta con un molex de alta fiabilidad para evitar falsos contactos con los cables evitando lecturas erróneas de las señales.

La etapa de potencia está separada de la etapa lógica (ver figura 2.21), ambas cuentan con su propia masa, dichas masas solo se unen a través de una delgada línea para evitar que las corrientes parásitas del motor dañen o alteren las señales lógicas.

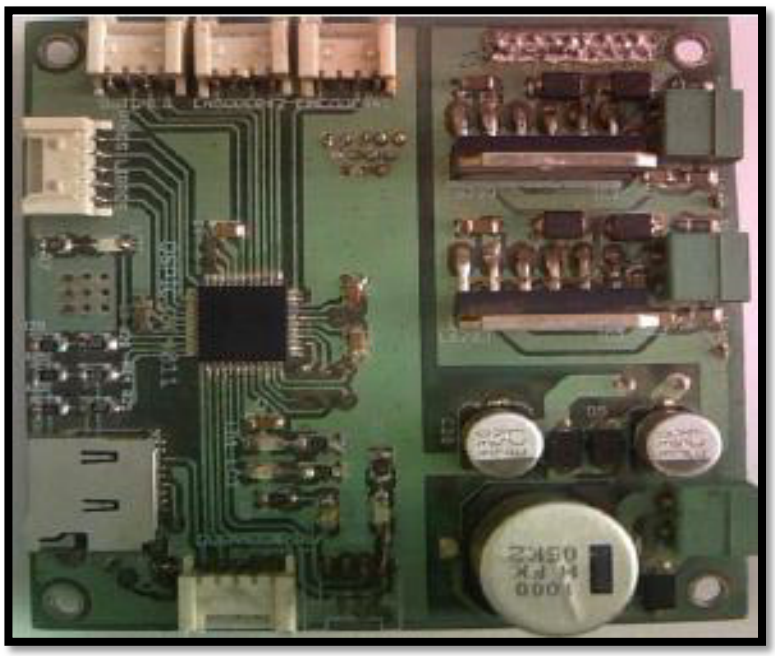


Figura 2.21.- Tarjeta electrónica implementada.

Capítulo III
Sistema de Control del Robot

El sistema de control del robot móvil está compuesto por cuatro etapas fundamentales y complementarias: lectura de encoders, análisis odométrico, controlador y desplazamiento del robot, tal como se muestra en la figura 3.1. Cada etapa conforma una capa de programación en cuyos algoritmos emplean las capas anteriores. Además esta separación de los subprocesos permite una programación de los algoritmos de forma estructurada, lo cual hace posible la reutilización de código, rápida detección de fallas o errores, así como una mejor presentación y documentación. Una de las principales ventajas de la programación de algoritmos en capas es la facilidad de desarrollar y actualizar cada etapa por separado, mejorando la versión continuamente sin cambiar la estructura global del código. Esto conlleva a la creación de librerías y sub-librerías que se interconectan entre sí para estructurar los bloques ya mencionados.

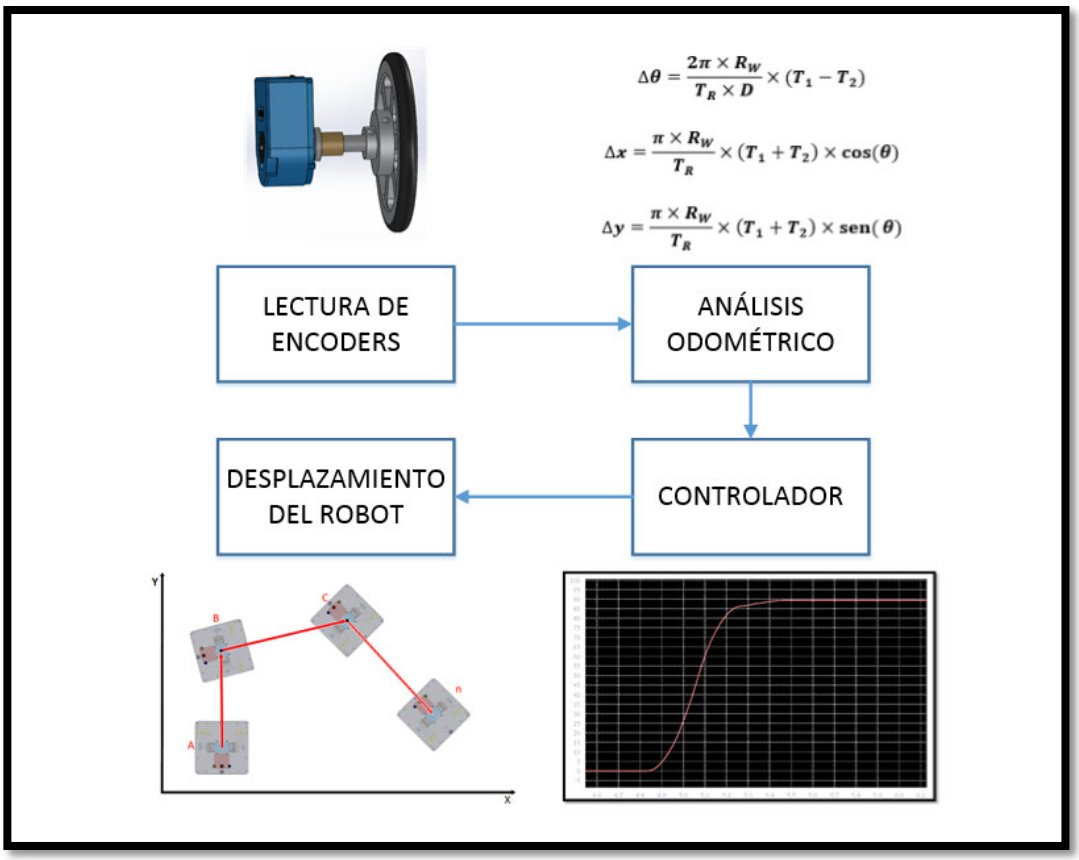


Figura 3.1.- Elementos del sistema de control del Robot.

1. Lectura de encoders

El sistema de lectura de los encoders que se encuentran ubicados en los extremos de la estructura del robot móvil registran el movimiento real del robot generado por la locomoción de los motores. Como se observa en la figura 3.2, cuando el robot se desplaza, los encoders detectan el movimiento a través de la rotación de sus ruedas y generan un tren de pulsos llamados Ticks (ver Capítulo 1, 2.Encoder). Los pulsos son leídos por el dsPIC a través de sus entradas digitales, éste a su vez ejecuta un algoritmo que procesa las señales del tren de pulso, calculando y acumulando los ticks. Los cuáles serán empleados posteriormente para el cálculo de las coordenadas y orientación angular del robot.

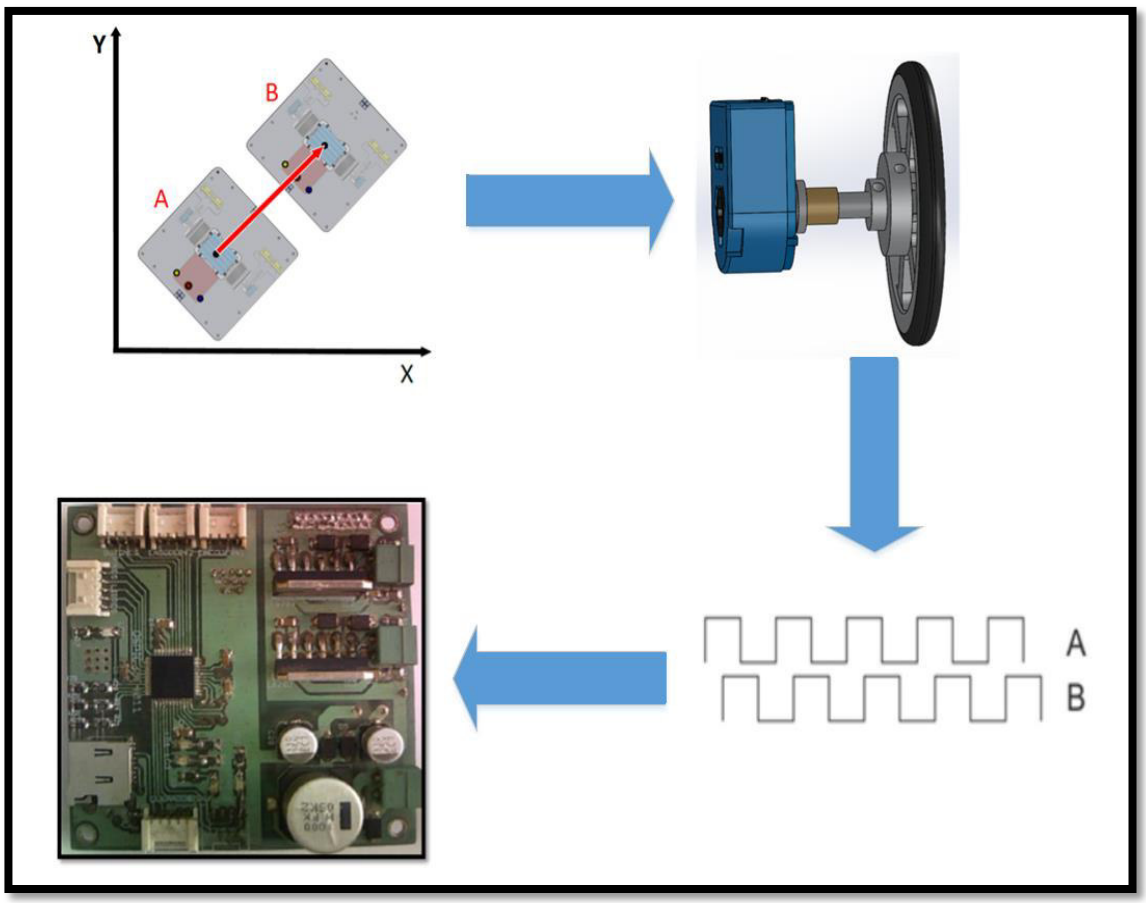


Figura 3.2.- Diagrama del sistema de lectura de encoders.

La lectura de las señales de los encoders se basa en la detección y acumulación de los ticks. Según el capítulo 1, 2.Encoder, se observó que el tick es generado al obtener los 4 estados (01 00 10 11) uniendo los bits de los canales A y B. Esta secuencia de bits es cíclica y repetitiva tal como se muestra en la figura 1.7 del capítulo 1.

Las secuencias de bits (AB) generadas al desplazar el encoder en sentido anti horario como horario están ordenados y presentados en la Tabla 3.1. Empezaremos analizando el sentido anti-horario, se observa que al convertir los bits (AB) del sistema binario al decimal presentan un indicio de secuencia (1 0 2 3 1 0), el cual consiste en un descenso en el valor. Para que la secuencia decremente en forma ordenada se hará un intercambio numérico entre el 2 y el 3, obteniendo la secuencia (1 0 3 2 1 0). Se empleó el mismo análisis para el sentido horario, en la secuencia original convertida al sistema decimal (3 2 0 1 3) se intercambió de igual manera el 2 por 3, obteniendo una secuencia que incrementa de forma ordenada (2 3 0 1 2 3).

← **Sentido Anti horario (AB)**	**Decimal**	**Intercambio de 2 por 3**	→ **Sentido Horario (AB)**	**Decimal**	**Intercambio de 2 por 3**
0 1	1	1	1 1	3	**2**
0 0	0	0	1 0	2	**3**
1 0	2	3	0 0	0	**0**
1 1	3	2	0 1	1	**1**
0 1	1	1	1 1	3	**2**
0 0	0	0	1 0	2	**3**
1 0	2	3	0 0	0	**0**
1 1	3	2	0 1	1	**1**

Tabla 3.1.- Secuencia de bits generados por el encoder.

Resumiendo lo observado en la tabla anterior, cuando el eje del encoder rota en sentido anti-horario la secuencia numérica disminuye cíclicamente; en sentido horario la secuencia aumenta. Este comportamiento se muestra en la figura 3.4.

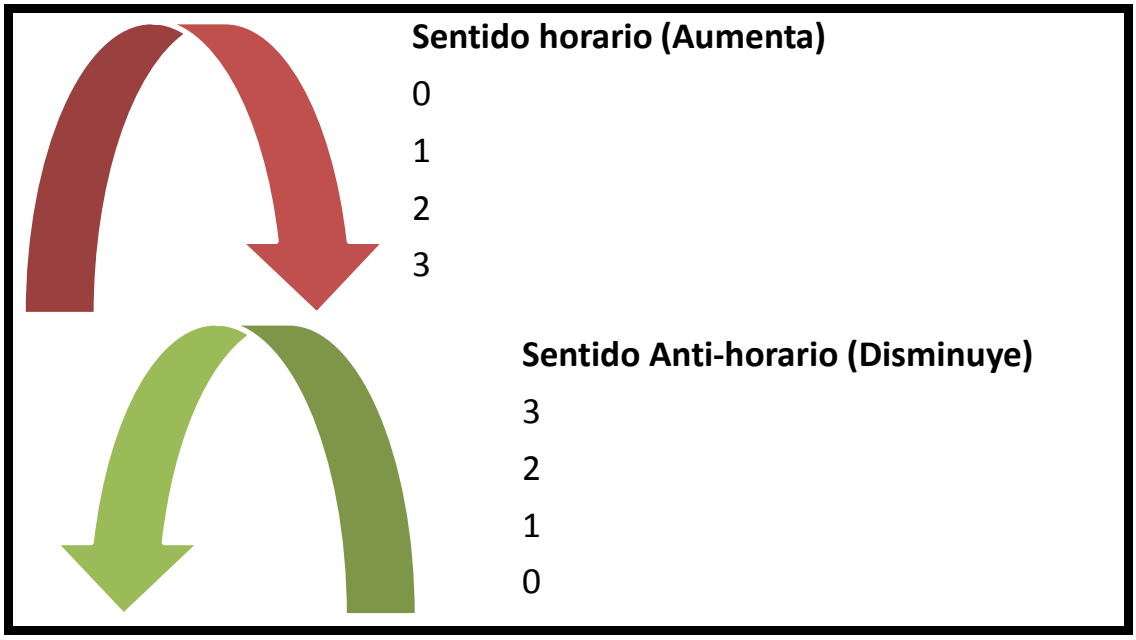


Figura 3.4.- Secuencias generadas.

Una vez identificadas las secuencias numéricas en función al sentido de giro del encoder, se desarrolló un algoritmo capaz de contabilizar los ticks. El esquema del algoritmo se muestra en la figura 3.5, éste consiste en aumentar o disminuir el valor de una variable llamada "contador", esta variable es inicializada con un valor de 4 para evitar trabajar con números negativos, ya que la variable podrá disminuir su valor cuatro veces hasta llegar a 0 cuando se trate de un tick en sentido anti-horario o aumentar su valor cuatro veces hasta llegar a 8 cuando se trate de un tick en sentido horario.

Cada vez que el valor numérico del estado actual de la secuencia del tick (0 1 2 3) aumente en una unidad (por ejemplo de 0 a 1, 1 a 2, 2 a 3 o 3 a 0) significará que el encoder se desplazó en sentido horario y se incrementará el valor de la variable "contador" en 1. Cuando el valor numérico del estado actual disminuya en una unidad (por ejemplo de 3 a 2, 2 a 1, 1 a 0 o 0 a 3) significará que el encoder se desplazó en sentido anti-horario y se disminuirá el valor de la variable "contador" en 1.

Quiere decir que cuando la variable "contador" llegue a 8 habrá contado los cuatro estados de un tick en sentido horario e incrementará en 1 la cantidad de ticks acumulados; cuando llegue a 0 habrá contado los cuatro estados de un tick en sentido anti horario y disminuirá en 1 la cantidad de ticks [14].

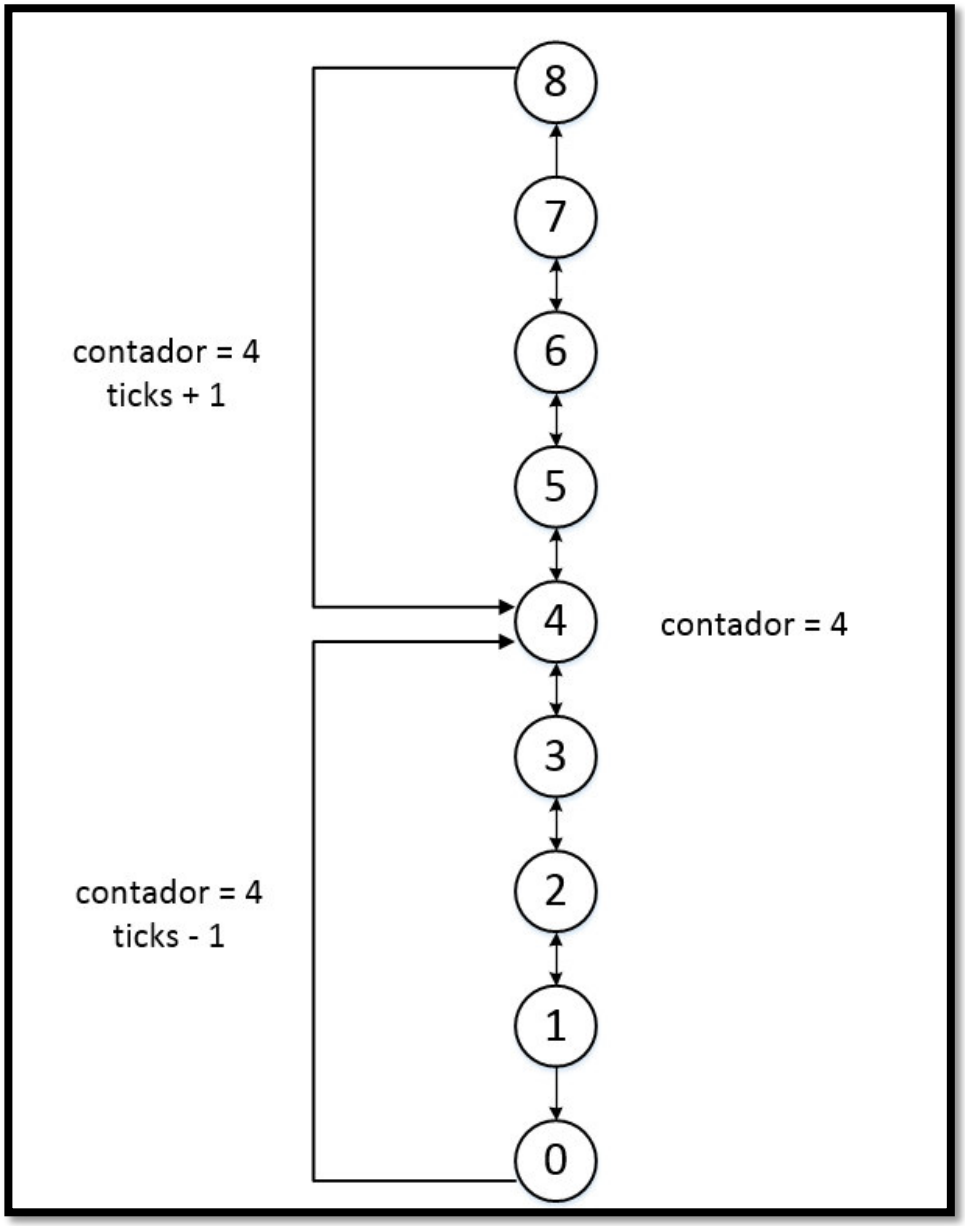


Figura 3.5.- Esquema del algoritmo de contaje de ticks.

El diagrama de flujo del algoritmo completo de la lectura de encoder se observa en la figura 3.6. La primera acción es la inicialización de las variables "a", "b", "digito", "digito_ant" y "contador". Todas estas variables se inicializan en 0 excepto la variable "contador" que es instanciada en 4. Luego se le asigna el valor de la señal digital de los canales A y B a las variables "a" y "b". Con los valores binarios de "a" y "b" se forma un número decimal que es asignado a la variable "digito" (entero de 16 bits), con esta variable se hará la comparación para detectar si el eje se movió en sentido horario o anti horario. El paso siguiente es el intercambio numérico entre el número 2 y 3 para generar una secuencia ordenada y coherente. Esta operación consiste en preguntar si el valor de "digito" es 2 e intercambiarlo por 3; y viceversa. Para poder detectar el sentido del eje del encoder se ejecutarán dos condicionales tomando como referencia el valor anterior de "digito", el cual está asignado a la variable "digito_ant". La primera condición es si el valor de "digito" es igual al valor de "digito_ant" + 1 y "digito" es diferente de 0; o si "digito_ant" es igual a 3 y "digito" igual a 0. Esta primera condición hace la pregunta si el eje está girando en sentido horario ya que la cuenta de los estados del tick estaría aumentando, tal como se representó en la figura 3.4. Si esta primera condicional es verdadera se aumenta en uno el valor de "contador", caso contrario se preguntará la segunda condicional. Si el valor de "digito" es igual al valor de "digito_ant" - 1 y "digito" es diferente de 3; o si "digito_ant" es igual a 0 y "digito" igual a 3. Esta segunda condición hace la pregunta si el eje está girando en sentido anti horario debido a que la cuenta de los estados del tick estaría disminuyendo. Si se cumple esta condición se disminuye en 1 el valor de "contador". Finalmente se analiza el valor numérico de "contador", si vale 8 se aumenta en 1 la cantidad de ticks y se inicializa el valor de "contador" en 4; si vale 0 se disminuye en 1 la cantidad de ticks y se inicializa el valor de contador en 4. La última operación es asignarle el valor de "digito" a "digito_ant" para volver a ejecutar el algoritmo.

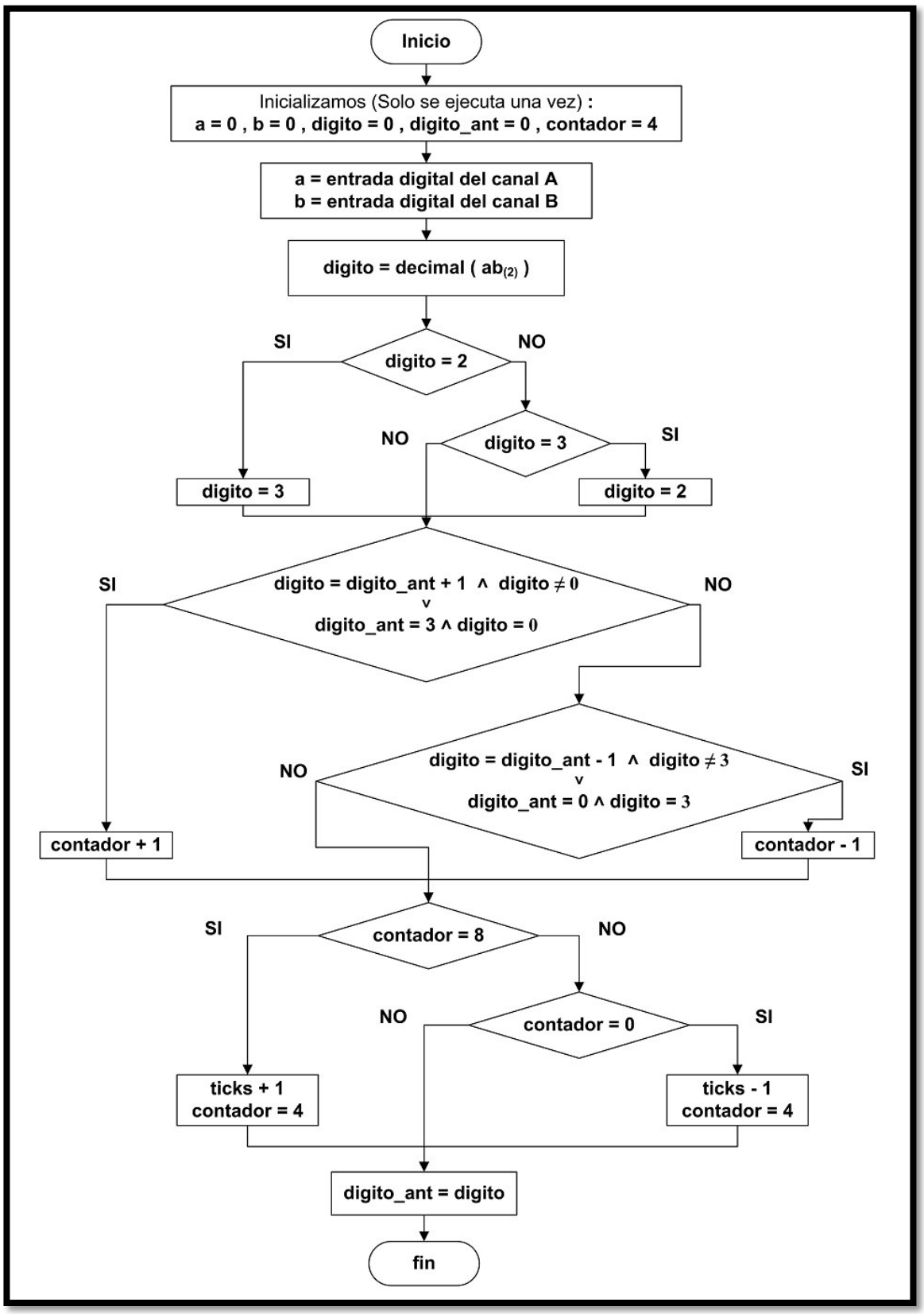


Figura 3.6.-Diagrama de flujo de lectura de encoder.

La implementación del algoritmo de lectura de encoder en el dsPIC se realizó en lenguaje C, creando una librería llamada "encoder".

Para crear la librería se emplearon los comandos de las líneas 1 y 2. Con estos comandos se define como un archivo header (.h).

```
1. #ifndef encoder_h
2. #define encoder_h
```

Como se mencionó anteriormente las librerías pueden tener sus propias variables globales, las cuales pueden ser usadas desde cualquier parte del programa, incluso desde otras librerías. En esta librería se crearán dos variables globales, "Ti" el contador de ticks del encoder de la izquierda y "Td" el contador de ticks del encoder de la derecha. Estas variables son declaradas e inicializadas en las líneas 3 y 4.

```
3. int Ti = 0;
4. int Td = 0;
```

El algoritmo mostrado en la figura 3.6 es genérico para la lectura de cualquier encoder incremental. En nuestro caso, el robot cuenta con dos encoders ubicados a la izquierda y derecha de la estructura. Para estructurar mejor la programación se crearon tres funciones, "muestreo_encoder_2()" para la lectura de los dos encoders, "muestreo_encoder_DER()" para la lectura del encoder de la derecha y "muestreo_encoder_IZQ()" para la lectura del encoder de la izquierda.

```
5. void muestreo_encoders_2(void);
6. void muestreo_encoder_DER(void);
7. void muestreo_encoder_IZQ(void);
```

La función "muestreo_encoders_2" llama a las otras dos funciones para tener una única función de muestreo.

```
8. void muestreo_encoders_2(void)
{
9. muestreo_encoder_DER();
10. muestreo_encoder_IZQ();
}
```

La función "muestreo_encoders_DER" es la implementación exacta del algoritmo descrito en la figura 3.6. En las líneas 12 y 13 se declaran las variables "a", "b" y "digito", estas variables son del tipo int debido a que trabajarán con números enteros. En cambio en las líneas 14 y 15 se declaran "digito" y "digito_ant" como variables estáticas, quiere decir que esas líneas solo se ejecutarán una sola vez durante toda la ejecución del programa, aun si la función es llamada varias veces. Se utilizó este tipo de declaración debido a que esas variables tienen memoria, quiere decir que su valor depende de su estado anterior, por tal motivo no pueden ser inicializadas cada vez que se ejecuta la función.

```
11.    void muestreo_encoder_DER(void)
{
12.    int a = 0, b = 0;
13.    int digito = 0;
14.    static int digito_ant = 0;
15.    static int contador = 4;
```

La asignación del valor de la señal digital de los canales A y B se ejecuta a través de una condición tal como se muestra en las líneas 16, 17, 18 y 19. Si la lectura de la entrada digital asignada al canal vale 1, la variable valdrá 1 también; caso contrario valdrá 0.

```
16.    if(input_encoder_DA==1) a = 1;
17.    else a = 0;
18.    if(input_encoder_DB==1) b = 1;
19.    else b = 0;
```

Para formar el valor de la variable "digito" se hace un corrimiento a la izquierda de un bit a la variable "a" y se le suma "b" para que ocupe el lugar del bit que fue desplazado, de esta manera se forma el valor numérico decimal (ver línea 20). Luego en las líneas 21 y 22 se ejecuta el intercambio numérico de dígito cuando vale "2" o "3"

```
20.    digito = (a<<1)+b;

21.    if(digito==2)  digito = 3;
22.    else if(digito==3) digito = 2;
```

La detección del sentido de giro se realiza a través de dos condicionales, si se cumple la primera condición se aumenta en 1 el valor de "contador" cuando el eje gira en sentido horario (línea 23), caso contrario se disminuye en 1 en valor de "contador" cuando gira en sentido anti horario (línea 24).

```
23.   if( ( ( digito==digito_ant + 1 && digito != 0 ) ) || ( ( digito_ant==3 && digito==0 ) ) )
      contador++;
24.   else if( ( ( digito==digito_ant-1 && digito != 3 ) ) || ( ( digito_ant==0 && digito==3 ) ) )
      contador--;
```

Finalmente se pregunta si "contador" vale 8 o 0 para aumentar o disminuir la cuenta de "Td" (ticks del encoder de la derecha) respectivamente (líneas 25 y 26). Además se le asigna el valor de "digito" a "digito_ant".

```
25.   if(contador==8)  { Td += 1; contador = 4;}
26.   else if(contador==0) {Td -= 1; contador = 4;}

27.   digito_ant=digito;
}
```

La misma estructura de programación es desarrollada para la lectura del encoder de la izquierda con la función "muestreo_encoder_IZQ".

```
28.   void muestreo_encoder_IZQ(void)
{
29.   int a = 0,b = 0;
30.   int digito = 0;
31.   static int digito_ant = 0;
32.   static int contador = 4;

33.   if(input_encoder_IA==1) a = 1;
34.   else a = 0;
35.   if(input_encoder_IB==1) b = 1;
36.   else b = 0;

37.   digito = (b<<1) + a;
```

```
38.     if(digito==2) digito = 3;
39.     else if(digito==3) digito = 2;
40.     if ( ( ( digito == digito_ant + 1   && digito != 0 ) )  || ((digito_ant==3  &&  digito==0 ) ) )
        contador++;
41.     else if ((( digito == digito_ant - 1  && digito != 3 ) ) || ( ( digito_ant == 0  &&  digito == 3 )))
        contador--;

42.     if(contador == 8 )  { Ti += 1; contador = 4; }
43.     else if(contador == 0 )  { Ti -= 1;  contador = 4;}

44.     digito_ant = digito;
}

45.     #endif
```

La función "muestreo_encoders_2()" se ejecutará cíclicamente dentro de un temporizador de ciclo de interrupción de 1,2 us, quiere decir que cada este periodo de tiempo se ejecutará el algoritmo para determinar si el eje del encoder se desplazó y cuanto lo hizo.

2. Análisis Odométrico

El sistema requiere conocer la ubicación del robot durante su funcionamiento para poder ser controlado a través de una estrategia de control, las cuales serán explicadas posteriormente. En la figura 3.7 se muestran los parámetros que representan la ubicación del robot en el plano: coordenada X, coordenada Y y la orientación angular θ [15].

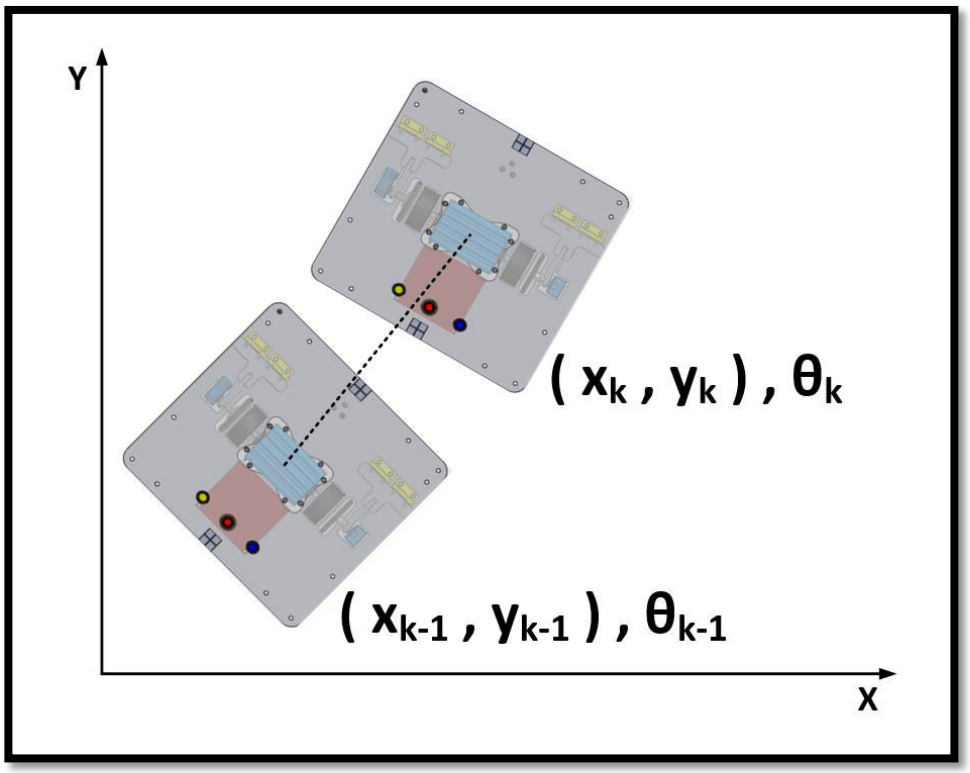


Figura 3.7.- Parámetros de ubicación del robot.

Para calcular la ubicación del robot móvil se utilizó la odometría (ver Capítulo 1, 3.Odometría). La cual estima los parámetros a partir de la información obtenida por sensores. En este proyecto de emplearon encoders incrementales para medir el desplazamiento de las ruedas del robot durante el movimiento. Esta estimación de parámetros se calcula a través de las ecuaciones deducidas en el marco teórico, donde las ecuaciones 1.14, 1.15 y 1.16 representan las variaciones de los parámetros.

$$\Delta\boldsymbol{\theta} = \frac{\boldsymbol{2\pi \times R_W}}{\boldsymbol{T_R \times D}} \times (\boldsymbol{T_i - T_d}) \ldots (\mathbf{1.14})$$

$$\Delta\boldsymbol{x} = \frac{\boldsymbol{\pi \times R_W}}{\boldsymbol{T_R}} \times (\boldsymbol{T_i + T_d}) \times \mathbf{cos}(\boldsymbol{\theta}) \ldots (\mathbf{1.15})$$

$$\Delta\boldsymbol{y} = \frac{\boldsymbol{\pi \times R_W}}{\boldsymbol{T_R}} \times (\boldsymbol{T_i + T_d}) \times \mathbf{sen}(\boldsymbol{\theta}) \ldots (\mathbf{1.16})$$

Donde:

R_W = Radio de la llanta del encoder.
D = Distancia entre las llantas del encoder.
T_i = Cantidad de ticks registrados por el encoder de la izquierda.
T_d = Cantidad de ticks registrados por el encoder de la derecha.
T_R = Cantidad total de ticks en una revolución

A partir de la variación de coordenadas y del ángulo de dirección se calculan los valores actuales de la posición y orientación con una ecuación en diferencias.

$$\boldsymbol{\theta_k} = \boldsymbol{\theta_{k-1}} + \Delta\boldsymbol{\theta} \ldots (\mathbf{1.17})$$

$$\boldsymbol{x_k} = \boldsymbol{x_{k-1}} + \Delta\boldsymbol{x} \ldots (\mathbf{1.18})$$

$$\boldsymbol{y_k} = \boldsymbol{y_{k-1}} + \Delta\boldsymbol{y} \ldots (\mathbf{1.19})$$

Estas ecuaciones serán implementadas dentro del dsPIC a través de un algoritmo, el cual se muestra en la figura 3.8.

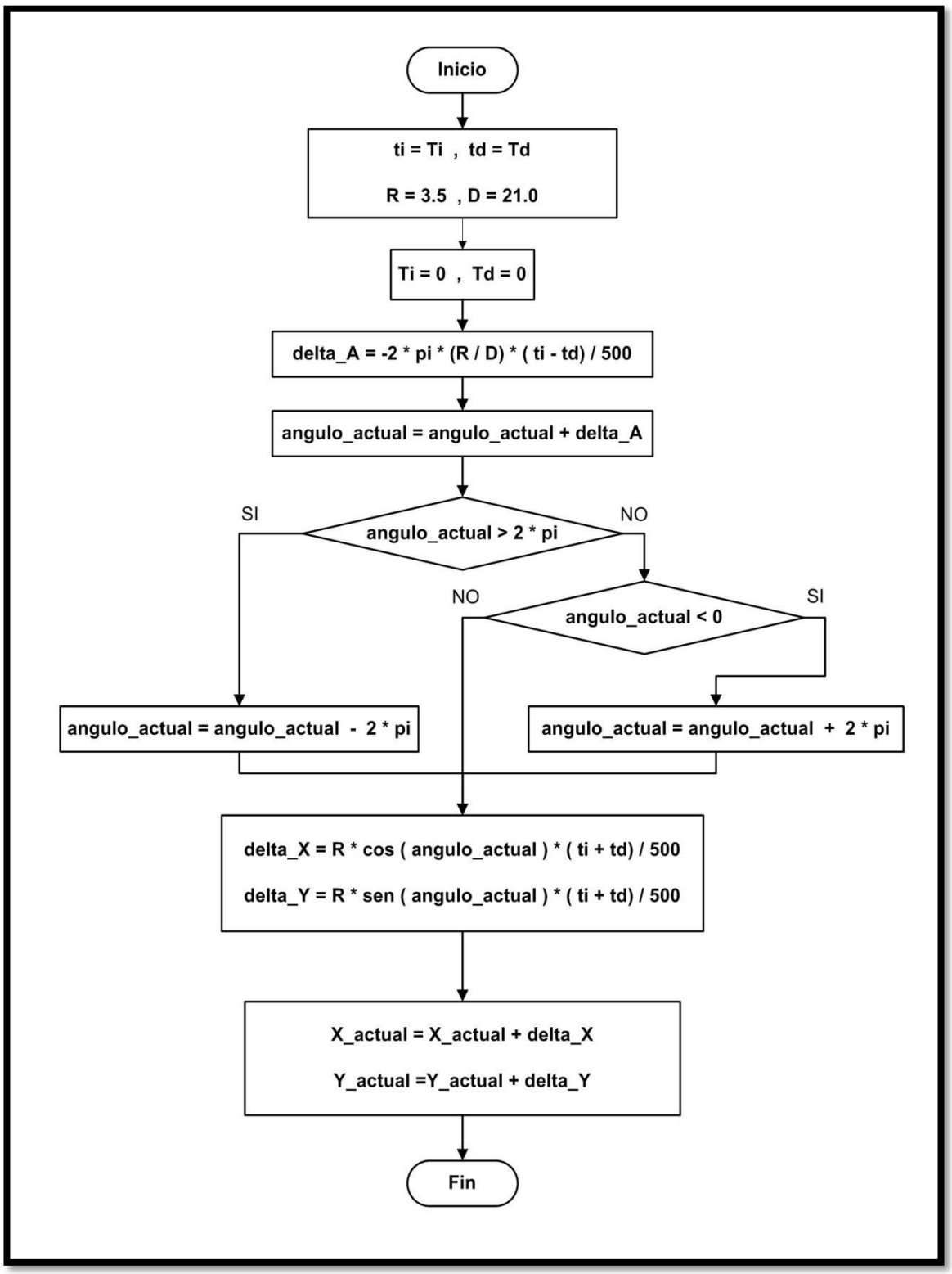


Figura 3.8.- Diagrama de flujo del análisis odométrico.

El algoritmo inicia declarando las variables "ti" (ticks del encoder izquierdo) y "td" (ticks del encoder derecho), estas variables son inicializadas con el valor de las variables globales de la librería encoder ("Ti" y "Td") que tienen la cuenta actualizada de los ticks generados por ambos encoders, también se declara e inicializa la variable "R" (radio de la rueda acoplado al eje del encoder) con 3.5 cm y "D" (distancia entre las ruedas de los encoders) con 21.0 cm. Luego se reinicia la cuenta de las variables globales "Ti" y "Td", para que se vuelva a contabilizar los ticks y no se acumulen por largos periodos de tiempo corriendo el riesgo de sobrepasar el límite del tipo de dato. La variación angular es calculada y asignada a la variable "delta_A", la cual es utilizada para calcular el "angulo_actual" aplicando una ecuación en diferencias de primer orden. Para trabajar solo en referencia de 0 a 360º de la posición angular del robot se hará una reducción a la primera vuelta. Quiere decir que si el ángulo actual es mayor a 360º o 2π se le reducirá a la primera revolución restándole 2π. Por otro lado si el ángulo actual es negativo o menor que 0 se adicionará 2π. De esta manera siempre el ángulo actual estará referenciado a la primera vuelta. Luego se calculan las variaciones de las coordenadas "X" e "Y" y se asignan a las variables "delta_X" y "delta_Y", ambos en función del ángulo actual y los contadores de ticks de cada encoder. Finalmente se calculan "X_actual" e "Y_actual" adicionándole a sus valores los respectivos diferenciales.

Una vez hallada la posición del robot se calculará la velocidad lineal del mismo. Para el cálculo de dicho parámetro se emplearon los resultados del algoritmo de análisis odométrico (coordenadas X e Y). El diagrama de flujo del algoritmo de cálculo de la velocidad instantánea lineal se observa en la figura 3.9. El algoritmo consiste en calcular la velocidad instantánea en el eje X "velocidad_inst_x" y la velocidad instantánea en el eje Y "velocidad_inst_y", estas variables se calculan dividiendo el "delta_X" y "delta_Y" entre el tiempo. Finalmente se calculará la velocidad instantánea "velocidad_inst" como la resultante de las dos velocidades vectoriales "velocidad_inst_x" y "velocidad_inst_y". La resultante se calcula extrayendo la raíz cuadrada de la suma de los cuadrados de las dos velocidades vectoriales.

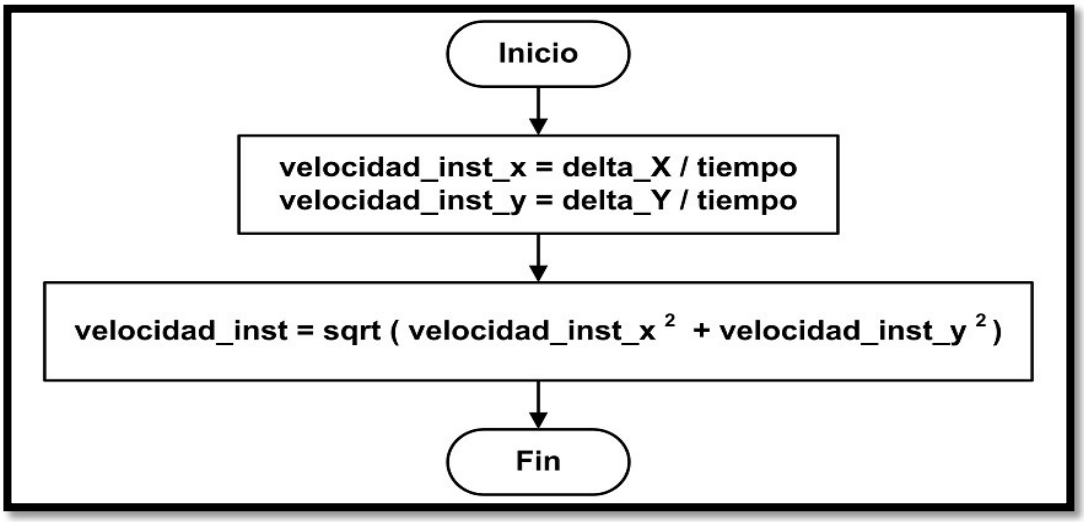


Figura 3.9.- Diagrama de flujo de cálculo de velocidad.

La implementación del algoritmo del cálculo de los parámetros cinemáticos del robot a través de la odometría en el dsPIC se realizó en lenguaje C, creando la librería "odometria".

Para crear la librería se emplearon los comandos de las líneas 1 y 2. Con estos comandos se define como archivo header (.h).

```
1. #ifndef odometria_h
2. #define odometria_h
```

En esta librería se declararán e inicialarán nueve variables globales. Los parámetros actuales ("angulo_actual", "X_actual" y "Y_actual"), los diferenciales ("delta_A", "delta_X" y "delta_Y") y las velocidades instantáneas ("velocidad_inst", "velocidad_inst_X" y "velociad_inst_Y") son inicializados en 0 excepto "angulo_actual" inicializado en 90º grados o pi/2.

```
3. float angulo_actual = pi/2.0;
4. float X_actual = 0.0;
5. float Y_actual = 0.0;
6. float delta_A = 0.0;
7. float delta_X = 0.0;
8. float delta_Y = 0.0;
9. float velocidad_inst = 0.0;
10. float velocidad_inst_x = 0.0;
11. float velocidad_inst_y = 0.0;
```

Como se observó en las figuras 3.8 y 3.9 se cuentan con dos algoritmos para la obtención de los parámetros cinemáticos. Por lo tanto se crearán dos funciones "muestreo_odometrico_posicion" y "muestreo_odometrico_velocidad". En la función de la velocidad se cuenta con un parámetro de entrada "tiempo" el cual define el tiempo transcurrido entre cada muestra.

```
12. void muestreo_odometrico_posicion ( void );
13. void muestreo_odometrico_velocidad ( float tiempo );
```

La función "muestreo_odometrico_posicion( )" ejecuta el algoritmo mostrado en la figura 3.7, este inicia declarando como tipo int las variables "ti" y "td" e inicializándolas con el valor de las variables globales "Ti" y "Td" (línea 15). Además se declaran las variables "R" y "D" (línea 16) como dato float e inicializan el valor de 3.5 (radio de la rueda acoplada al eje) y 21.0 (distancia entre las ruedas de los encoders).

```
14. void muestreo_odometrico_posicion()
{
15. int ti = Ti, td = Td;
16. float R = 3.50, D = 21.0;
```

Se reinicializa la cuenta de las variables globales de ticks de encoder de la derecha e izquierda (líneas 17 y 18).

```
17. Td = 0.0;
18. Ti = 0.0;
```

Luego se calcula "delta_A" aplicando la ecuación (3.1) para ser usado para calcular el "angulo_actual" (ver líneas 19 y 20).

```
19. delta_A = -2.0 * pi * (R / D)*((float) (ti - td)) / 500.0;
20. angulo_actual += delta_A;
```

Para que "angulo_actual" siempre tenga en valor de la primera vuelta se ejecutarán dos condicionales tal como se observa en las líneas 21 y 22.

```
21. if (angulo_actual > (2.0 * (float) pi))  angulo_actual -= 2.0 * ((float) pi);
22. else if (angulo_actual < 0)   angulo_actual += 2.0 * ((float) pi);
```

Se calculará los diferenciales de la posición de las coordenadas X "delta_X" e Y "delta_Y" en función a "angulo_actual" (líneas 23 y 24). Finalmente se calculan las variables globales "X_actual" y "Y_actual" adicionando los diferenciales a sus valores anteriores. Estas variables serán visibles por otras librerías que requieran tener la posición actual del robot (control y comunicación).

```
23. delta_X = R * cos(angulo_actual)*( (float) (ti + td))*((float) pi ) / 500.0;
24. delta_Y = R * sin(angulo_actual)*( (float) (ti + td))*((float) pi ) / 500.0;
25. X_actual += delta_X;
26. Y_actual += delta_Y;
}
```

El siguiente algoritmo implementado es el cálculo de la velocidad lineal instantánea del robot. Primero se hallan las velocidades instantáneas de los dos ejes y luego la componente resultante de ambas velocidades (ver líneas 28, 29 y 30).

```
27. void muestreo_odometrico_velocidad(float tiempo)
{
28. velocidad_inst_x = delta_X / tiempo;
29. velocidad_inst_y = delta_Y / tiempo;
30. velocidad_inst = sqrt(pow(velocidad_inst_x, 2) + pow(velocidad_inst_y, 2));
}

31. #endif
```

El periodo de ejecución de estas funciones es 10ms mediante una interrupción cíclica del temporizador del dspic.

3. Controlador

El objetivo del robot móvil es desplazarse de un punto a otro, para ello se deben ejecutar algoritmos de control para asegurar un error mínimo o aceptable durante la navegación [16]. El movimiento del robot inicia con la necesidad de desplazarse entre dos puntos, tal como se observa en la figura 3.10, donde el robot se encuentra en el punto P1 y requiere desplazarse hacia el punto P2.

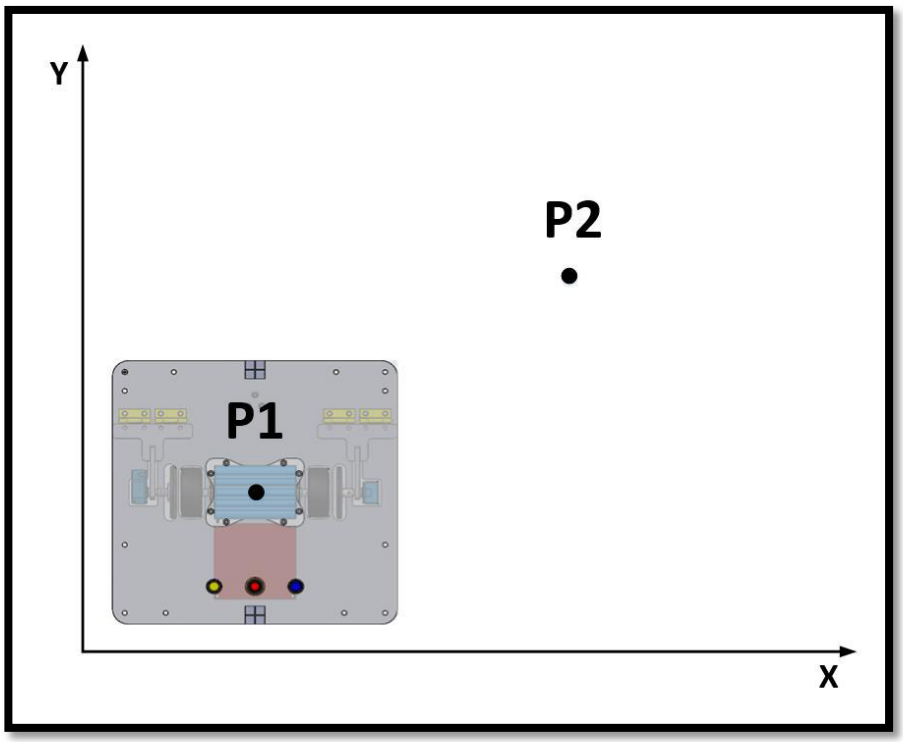


Figura 3.10.- Etapa de desplazamiento - 1.

Una vez identificado el punto al cual debe desplazarse, el robot trazará una recta entre ambos (ver figura 3.11) para determinar la orientación angular que seguirá. El robot deberá cambiar su orientación en dirección a la recta trazada, esto implica que gire sobre su eje hasta alinearse completamente. En la figura 3.12 se muestra el desplazamiento angular, donde el robot se alinea con la recta rotando hasta llegar al ángulo θ (ángulo de inclinación de la recta respecto al eje X). Para que este desplazamiento sea controlado y con un error aceptable se empleará un controlador PID de control angular, el cual se encargará de los movimientos de rotación.

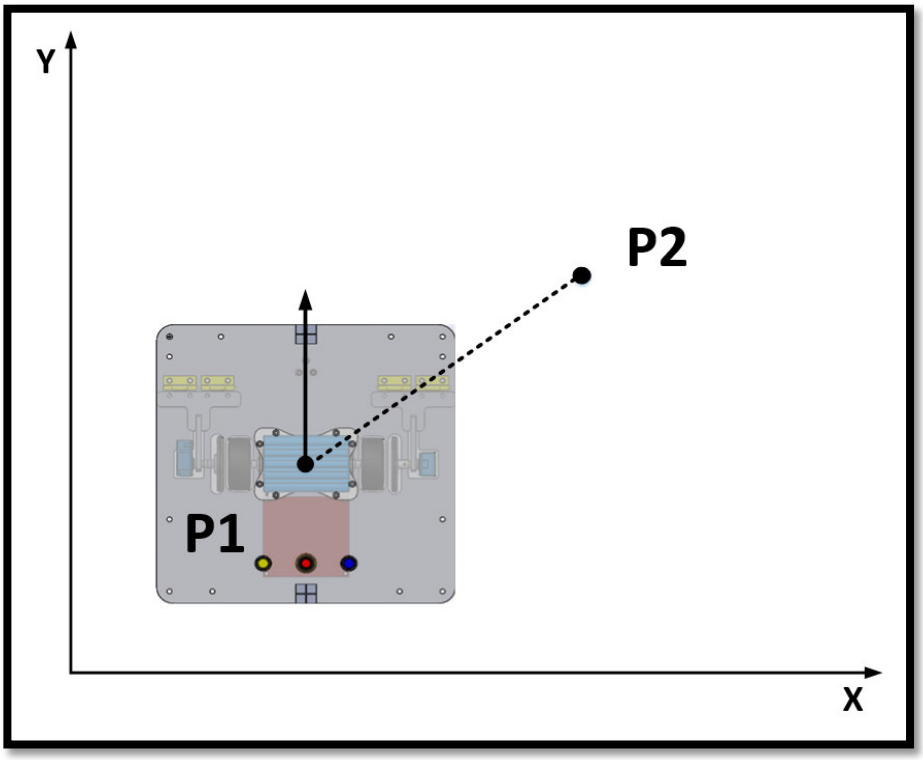


Figura 3.11.- Etapa de desplazamiento - 2.

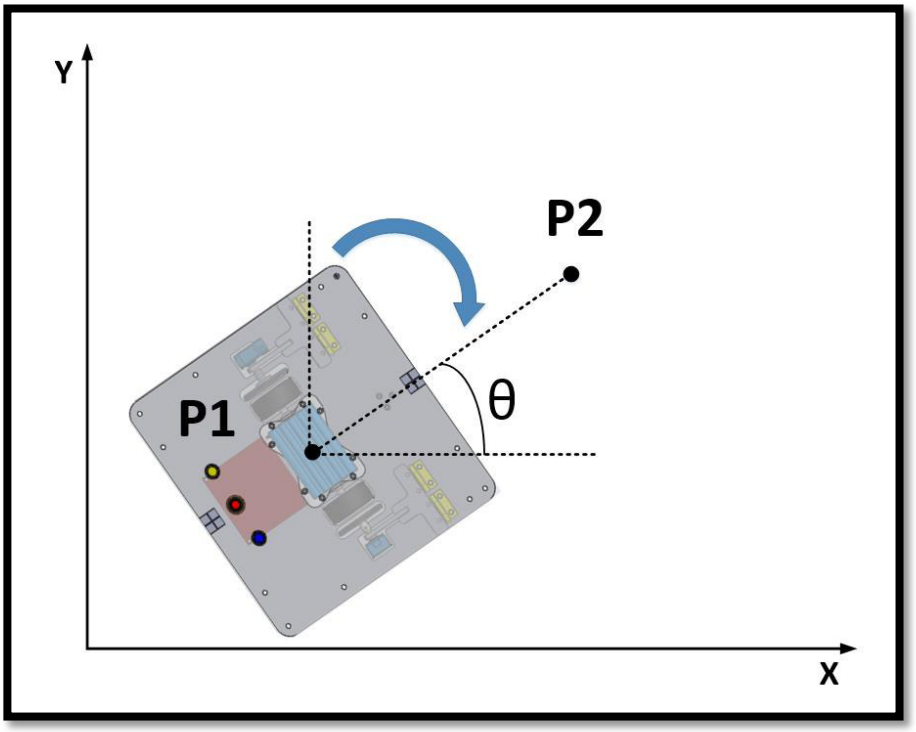


Figura 3.12.- Etapa de desplazamiento - 3.

Cuando el robot está alineado con la recta se inicia el desplazamiento lineal, el cual consiste en la navegación del punto a P1 al punto P2, tal como se muestra en la figura 3.13. Para controlar dicho movimiento se empleará un controlador PID de posición, el cual tendrá efecto cuando el robot requiera trasladarse de un punto a otro.

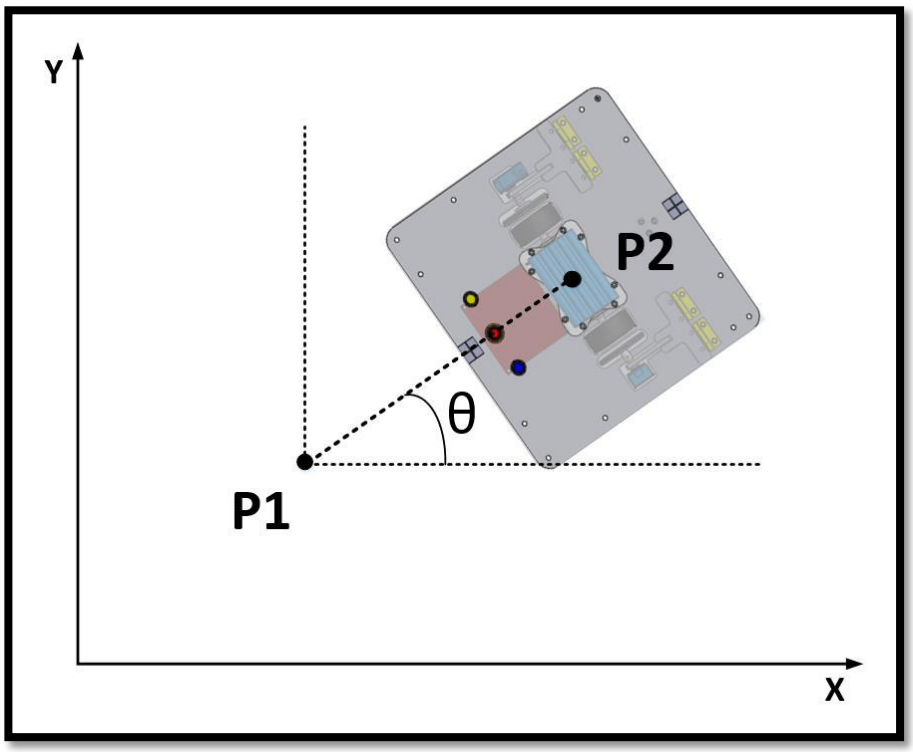


Figura 3.13.- Etapa de desplazamiento - 4.

Cada uno de estos controladores (PID de control angular y de posición) harán uso de los resultados obtenidos en el análisis odométrico. Ambos controladores PID manipularán la velocidad de los motores para modificar el valor de la orientación angular como el de la posición del robot a sus valores deseados. Antes de analizar los PID veremos cómo manipular la velocidad de la motores, así como el movimiento hacia adelante, atrás y giro hacia la derecha e izquierda.

3.1 Control de motores

El robot utiliza dos motores DC de 300 rpm alimentados con 12VDC. Estos se encargarán de los movimientos del robot (avanzar, retroceder, girar derecha y girar izquierda). Estos movimientos consisten en la conmutación de ambos motores que son controlados por el dsPIC a través del puente H (ver Capitulo 2, 2.3 Etapa de Potencia) ya que se requiere un driver para amplificar la corriente de la señal de control, el driver que se empleó para esta función fue el L6203 (ver Anexo A.3, L6203). Para controlar el sentido de giro de los motores el dsPIC genera cuatro señales de control, dos para cada motor tal como se muestra en la figura 3.13.

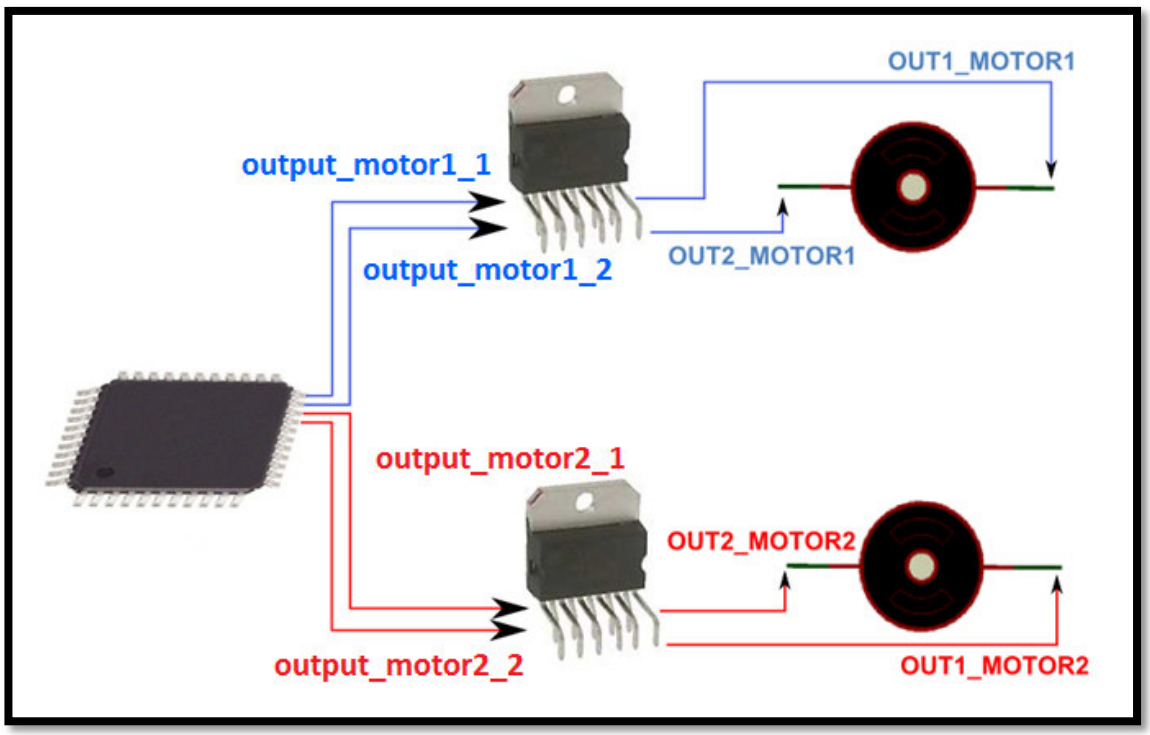


Figura 3.13.- Señales de conmutación de motores.

Para controlar cada motor se emplean dos señales digitales provenientes del dsPIC hacia el puente H, en función a los estados de los bits de dichas señales el motor girará en sentido horario o anti-horario. Si analizamos el movimiento de los dos motores con los que cuenta el robot se podrá definir con que combinaciones de señales el robot podrá avanzar, retroceder, girar a la izquierda y derecha. Para ordenar las señales que generan dichos tipos de movimientos se realizó la tabla 3.2. En esta tabla se muestran las cuatro señales "output_motor1_1", "output_motor1_2",

"output_motor2_1" y "output_motor2_2" indicando cada combinación para cada tipo de movimiento.

	Avanzar	Retroceder	Girar Derecha	Girar Izquierda
output_motor1_1	1	0	1	0
output_motor1_2	0	1	0	1
output_motor2_1	1	0	0	1
output_motor2_2	0	1	1	0

Tabla 3.2.-Combinacion de bits de movimiento.

Hasta este punto solo podemos controlar el sentido del movimiento del robot, mas no la velocidad del mismo ya que únicamente estamos enviando señales de mando para la conmutación de los motores. Para poder controlar la velocidad utilizaremos modulación por ancho de pulso (PWM). El PWM consiste en mantener un ciclo de periodo fijo de una señal cuadrada. A partir de ese periodo constante se modificará el tiempo que la señal se encuentra en 1 lógico, también conocido como ciclo de trabajo o duty cycle (ver figura 3.14). Este tiempo o duty cycle puede variar de un 0% hasta un 100%, las señales cuadradas por ejemplo trabajan al 50% ya que la mitad del tiempo del periodo están 1 lógico y la otra en 0 lógico.

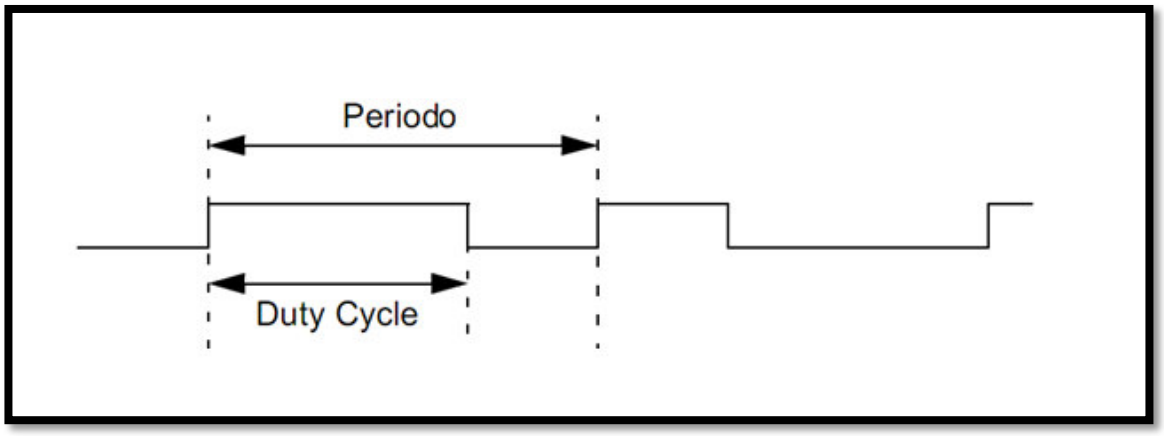


Figura 3.14.- Modulación por ancho de pulso.

Según el Anexo A.3 L6203, el driver o puente H cuenta con una entrada digital llamada "enable", cuando esta entrada está en 1 habilita las entradas para la conmutación del motor, quiere decir que el puente H trabajará según la tabla 3.2. Cuando la entrada "enable" está en 0 deshabilita las entradas de conmutación y la salida a los motores será 0 voltios. Por tanto, se utilizará esta entrada "enable" para modificar la velocidad del motor aplicándole una señal de PWM. Esta señal modulada generará un Vdc proporcional al duty cycle, quiere decir que a mayor ancho de pulso el Vdc será mayor lo que implicará una mayor velocidad del motor. Se necesitará una alta frecuencia en la señal modulada para evitar el ruido del motor en el rango audible [17]. Por lo que se escogió 20 KHz, ya que es la frecuencia más usada para control de motores DC con PWM. El motor recibirá un pulso de activación y otro de desactivación a una alta velocidad dando la impresión de reducir su velocidad o aumentar la misma cuando se reduce o aumenta el duty cycle de 0 a 100%. El dsPIC cuenta con un módulo especializado para la salida de PWM, este módulo fue configurado para tener una salida de frecuencia de 20KHz. Se realizaron pruebas con un osciloscopio para verificar el duty cycle y el periodo de la señal. Se obtuvieron los resultados mostrados en la figura 3.15 (duty cycle de 50%) y la figura 3.16 (duty cycle de 100%).

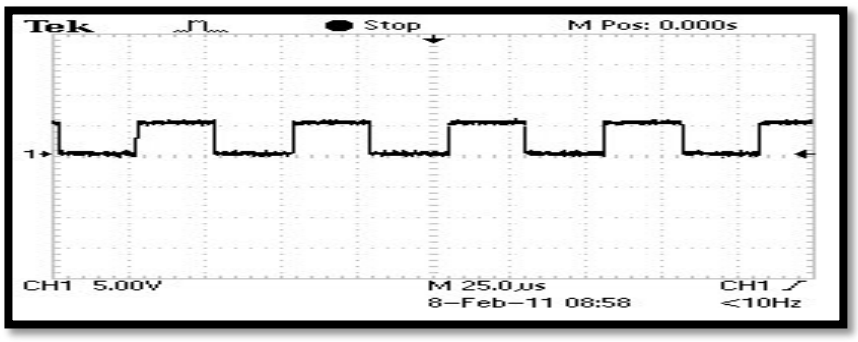


Figura 3.15.- Señal modulada con duty cycle a 50%.

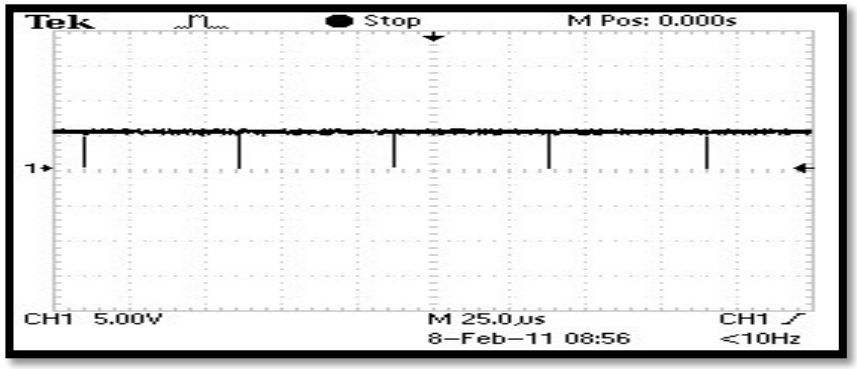


Figura 3.16.- Señal modulada con duty cycle a 100%.

Para uniformizar el control del sentido y velocidad del motor se creó una función que controle dichos parámetros. Esta función tiene como parámetro de entrada a "porc_pwm", el cual indica el porcentaje de velocidad del motor de 0 a 100%, además dependiendo del signo de la variable "porc_pwm" el motor girará en sentido horario o anti-horario pero al mismo porcentaje de velocidad. El diagrama de flujo del algoritmo de la función "motor" se muestra en la figura 3.17.

motor(porc_pwm)

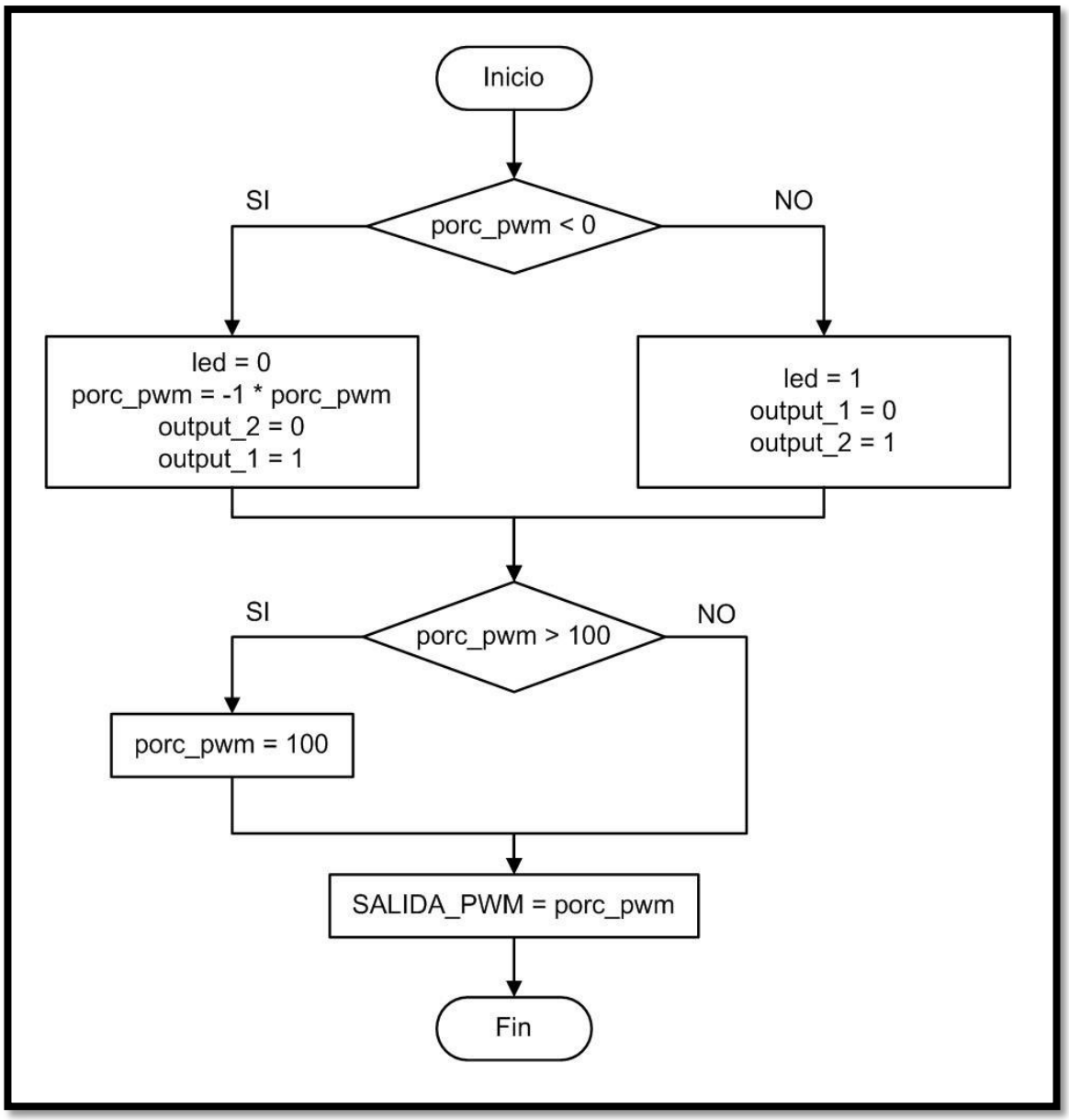


Figura 3.17.- Diagrama de flujo de la función motor.

Este algoritmo inicia preguntando si la variable de entrada "porc_pwm" es negativa o positiva. Si la variable es negativa se apaga el led que indica el sentido de giro, se multiplica la variable por "-1" para convertirla en positiva y finalmente se hace la conmutación de señales para que el motor gire en un sentido. Si la variable "porc_pwm" es positiva se activa el led y se conmuta las señales para que gire en el otro sentido. Además se necesita implementar un limitador de porcentaje de velocidad ya que el rango es únicamente de 0 a 100%. Si "porc_pwm" es mayor a 100, "porc_pwm" será igual a 100. Finalmente la salida de PWM del dsPIC se iguala a "porc_pwm".

Un punto resaltante respecto al control del movimiento de los motores es que no todo el rango de porcentaje de duty cycle genera movimiento en los motores. Por ejemplo un duty cycle de 5% debería suponer un 5% de la velocidad del motor, pero en realidad el motor no se moverá debido a que no romperá la inercia. Se hizo un registro de datos para saber a partir de qué porcentaje de duty cycle realmente el motor se desplaza. Estos datos fueron compilados y se muestran en la figura 3.18. Se encontró que el robot empieza a desplazarse cuando el motor recibe un duty cycle de 60%, debido a que requiere una fuerza inicial para romper la inercia propia del robot.

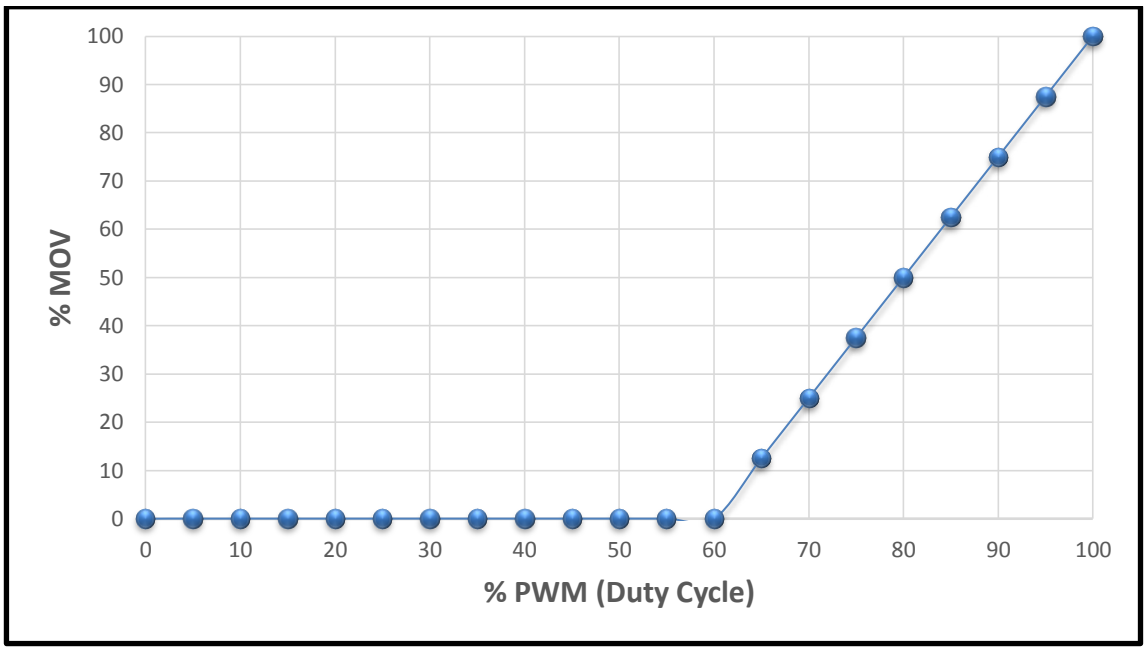


Figura 3.18.- Gráfica de porcentajes de duty cycle.

A partir de la gráfica de la figura 3.18 se obtiene una relación entre el porcentaje de PWM deseado (porc_pwm) y el porcentaje real de pwm (porc_pwm_real) con las ecuaciones 3.1 y 3.2 cuando porc_pwm es positivo y negativo respectivamente.

$$\boldsymbol{porc\_pwm\_real = 0.4 \times porc\_pwm + 60} \quad \ldots (\mathbf{3.1})$$

$$\boldsymbol{porc\_pwm\_real = 0.4 \times porc\_pwm - 60} \quad \ldots (\mathbf{3.2})$$

El algoritmo de conversión de porcentaje PWM se muestra en la figura 3.19.

porcent_real_mov(porc_izq_pwm, porc_der_pwm)

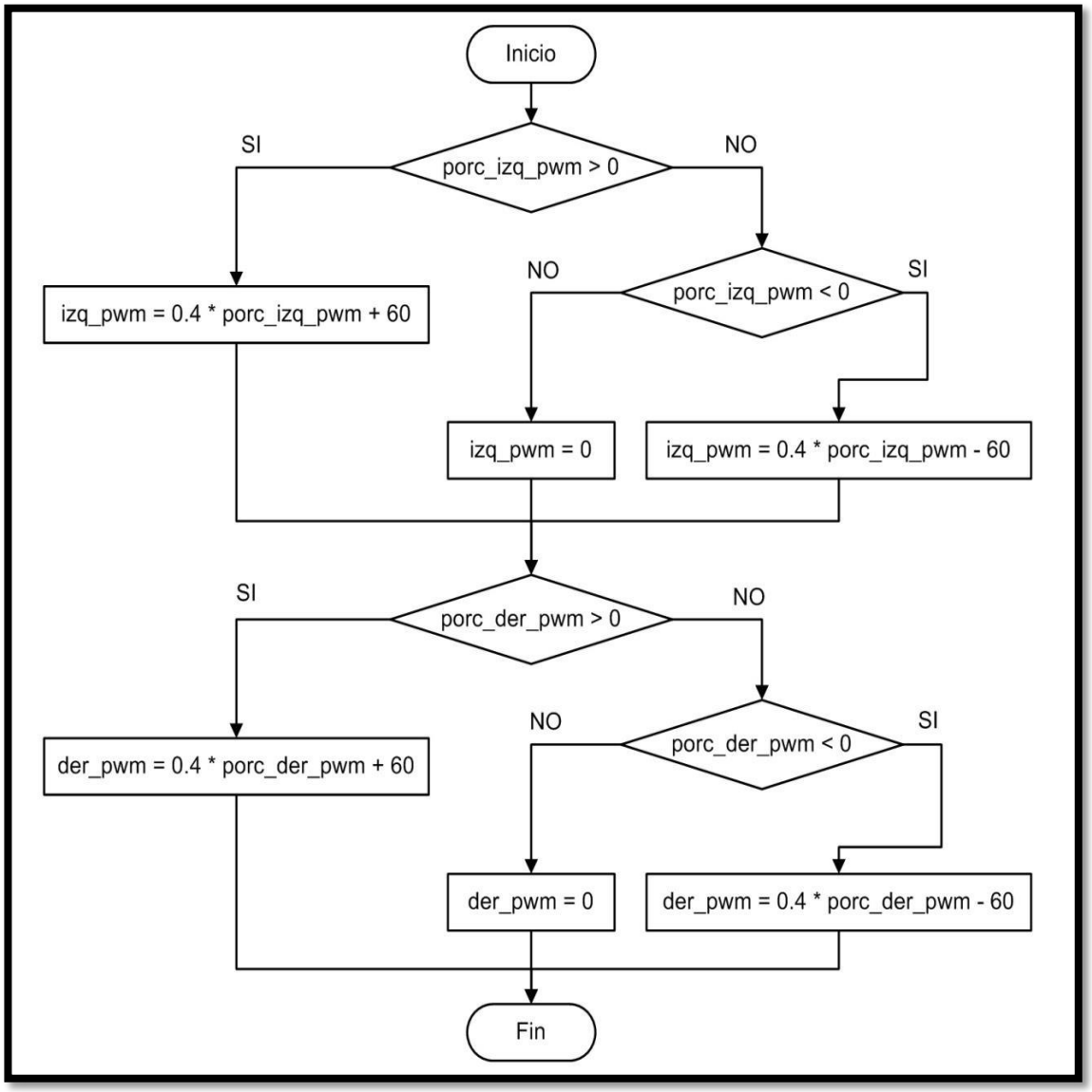


Figura 3.19.- Diagrama de flujo de la función porcent_real_mov.

El algoritmo consiste en analizar los porcentajes de PWM de los dos motores (izquierda y derecha) y realizar un escalamiento empleando las ecuaciones 3.1 y 3.2. Se inicia con el porcentaje de PWM del motor de la izquierda "porc_izq_pwm", si esta variable es mayor que cero (positiva) se emplea la ecuación 3.1, donde el movimiento real se asigna a la variable global "izq_pwm". Si la variable es menor a 0 (negativa) se ejecuta la ecuación 3.2 asignándola a "izq_pwm". Caso contrario "izq_pwm" se iguala a 0.

Luego se analiza el porcentaje de PWM del motor de la derecha "porc_der_pwm". Si esta variable es mayor que cero se emplea la ecuación 3.1, donde el movimiento real se asigna a la variable global "der_pwm". Si la variable es menor a 0 se ejecuta la ecuación 3.2 asignándola de igual manera a "der_pwm". Caso contrario "der_pwm" se iguala a 0.

Finalmente se creará la función "motores", la cual contiene los dos análisis anteriores. Esta función controlará ambos motores por lo que tendrá dos parámetros de entrada "porc_izq_pwm" y " porc_der_pwm". Primero se hará la transformación de PWM con la función "porcent_real_mov". Luego se utilizará la función "motor" para la derecha e izquierda ingresando como parámetro de entrada los porcentajes de PWM modificados. El diagrama de flujo de este algoritmo se observa en la figura 3.20

motores(porc_izq_pwm , porc_der_pwm)

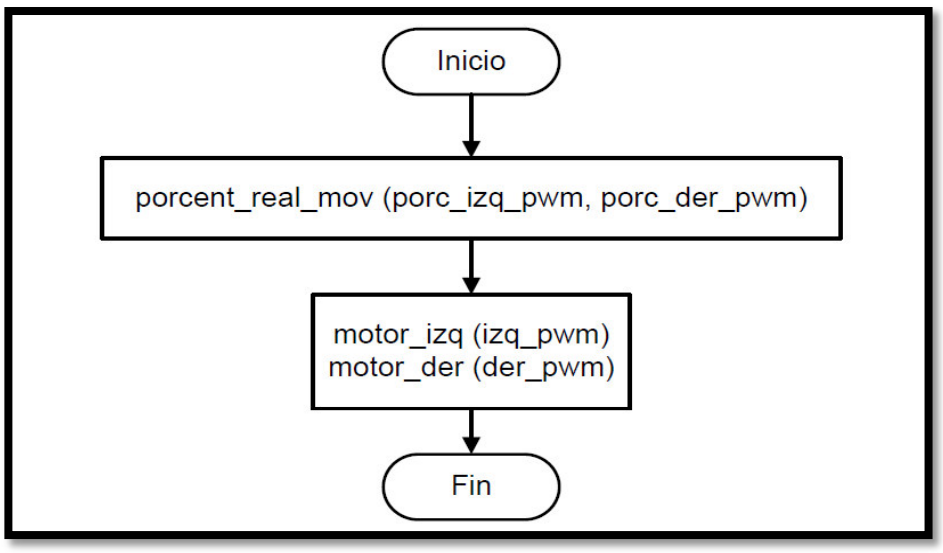


Figura 3.20.- Diagrama de flujo de la función motores.

La codificación de estos algoritmos se observa a continuación. Empezaremos configurando el módulo PWM del dsPIC basándonos en los registros del datasheet del módulo [18]. En la línea 1 se habilita la función de PWM del dsPIC colocando el bit del registro en 1. El robot posee dos motores, por lo que requerirá dos salidas independientes de PWM, en la líneas 2 y 3 de habilitan las salidas de PWM1 y PWM2. Además se requiere definir la frecuencia de la señal modulada, este parámetro se ajusta en la línea 4 a 20000 Hz o 20KHz. Finalmente se inicializan en 0 los registros que controlan el duty cycle en las líneas 5 y 6.

```
1.  PTCONbits.PTEN = 1;
2.  PWMCON1bits.PEN2H = 1;
3.  PWMCON1bits.PEN1H = 1;
4.  PTPER = 20000;
5.  PDC1 = 0;
6.  PDC2 = 0;
```

Se declaran las funciones "motor_izq", "motor_der", "motores" y "porcent_real_mov" (ver líneas 7, 8, 9 y 10). Las funciones "motor_izq" y "motor_der" son las implementaciones de la función "motor" para el motor de la izquierda y derecha.

```
7.  void motor_izq ( float porc_pwm );
8.  void motor_der ( float porc_pwm );
9.  void motores ( float porc_izq_pwm, float porc_der_pwm );
10. void porcent_real_mov ( float porc_izq_pwm, float porc_der_pwm );
```

Luego se declaran e inicializarán las variables globales que se utilizarán en esta librería como en otras. Cabe resaltar que "der_pwm" y "izq_pwm" (ver líneas 13 y 14) son las variables que se asignarán a las velocidades modificadas en la función "porcent_real_mov".

```
11. float porc_der_pwm = 0;
12. float porc_izq_pwm = 0;

13. float der_pwm = 0;
14. float izq_pwm = 0;
```

La función "motor_izq" ejecuta la misma lógica que la función "motor" (figura 3.17), a diferencia que esta función es dedicada para el motor de la izquierda.

En la línea 16 se realiza la condicional si "porc_pwm" es menor que cero. Si es menor se desactiva la salida del led 2 "output_led2". Además se multiplica por -1 el

valor de "porc_pwm" para volverlo positivo y se colocan los bits de conmutación según la tabla 3.2. Si el valor de "porc_pwm" es positivo se activa el led 2 e invierten los bits de conmutación. En la línea 25 se implementa el limitador, cuando el valor de "porc_pwm" es mayor a 100, este se volverá igual a 100. Finalmente se asigna el valor de "porc_pwm" al registro (PDC1) que controla la salida por PWM (ver línea 26).

```
15.     void motor_izq( float porc_pwm )
        {
16.         if ( porc_pwm < 0 )
            {
17.             output_led2 = 0;
18.             porc_pwm *= -1;
19.             output_motor1_1 = 0;
20.             output_motor1_2 = 1;
            }
21.         else
            {
22.             output_led2 = 1;
23.             output_motor1_2 = 0;
24.             output_motor1_1 = 1;
            }

25.         If ( porc_pwm > 100 )  porc_pwm = 100;

26.         PDC1 = ( int )( porc_pwm );
        }
```

De igual forma se implementará la función de control del motor de la derecha "motor_der".

```
27.     void motor_der ( float porc_pwm )
        {
28.         if ( porc_pwm < 0 )
            {
29.             output_led2 = 0;
30.             porc_pwm *= -1;
31.             output_motor2_1 = 0;
32.             output_motor2_2 = 1;
            }
33.         else
            {
34.             output_led2 = 1;
35.             output_motor2_2 = 0;
36.             output_motor2_1 = 1;
            }

37.         If ( porc_pwm > 100 )  porc_pwm = 100;

38.         PDC2 = ( int )( porc_pwm );
        }
```

En la línea 39 se implementa la función "porcent_real_mov" la cual realiza un escalamiento según el algoritmo de la figura 3.19.

```
39.     void porcent_real_mov( float porc_izq_pwm, float porc_der_pwm )
        {
40.             if ( porc_izq_pwm > 0 )  izq_pwm = 0.4 * porc_izq_pwm  +  60.0;
41.         else if ( porc_izq_pwm < 0 )  izq_pwm = 0.4 * porc_izq_pwm  -  60.0;
42.         else  izq_pwm = 0;

43.             If ( porc_der_pwm > 0 ) der_pwm = 0.4 * porc_der_pwm  +  60.0;
44.         else if ( porc_der_pwm < 0 ) der_pwm = 0.4 * porc_der_pwm  -  60.0;
45.         else der_pwm = 0;
        }
```

Finalmente se codifica la función "motores" (línea 46) la cual reúne a las funciones anteriores según la figura 3.20.

```
46.     void motores ( float porc_izq_pwm, float porc_der_pwm )
        {
47.         porcent_real_mov(porc_izq_pwm, porc_der_pwm);

48.         motor_izq(izq_pwm);
49.         motor_der(der_pwm);
        }

        }
50.     #endif
```

3.2 Control angular

El control angular buscará igualar la orientación angular del robot a la orientación deseada. En la figura 3.21 se observa que el robot inicialmente se encuentra con una orientación de θ1, se requiere que se oriente a un ángulo θ2. El controlador PID (ver Capítulo 1, 4.Controlador PID) deberá manipular la velocidad y el sentido de los motores con el fin de alinearse con un mínimo error aceptable.

El diagrama de bloques del sistema de control angular se muestra en la figura 3.22. Este posee un Setpoint (SP) o referencia, el cual es el ángulo al que debe orientarse el robot. El Setpoint se comparará con el valor actual del ángulo de orientación (θ) a través de una diferencia, dando como resultado un error (e). El error

ingresará como variable de entrada al controlador PID. El resultado del algoritmo del PID será la señal de control U, la cual se transformará a una salida de PWM para luego ser transmitidas hacia los motores, los cuales generarán una variación en la orientación del robot.

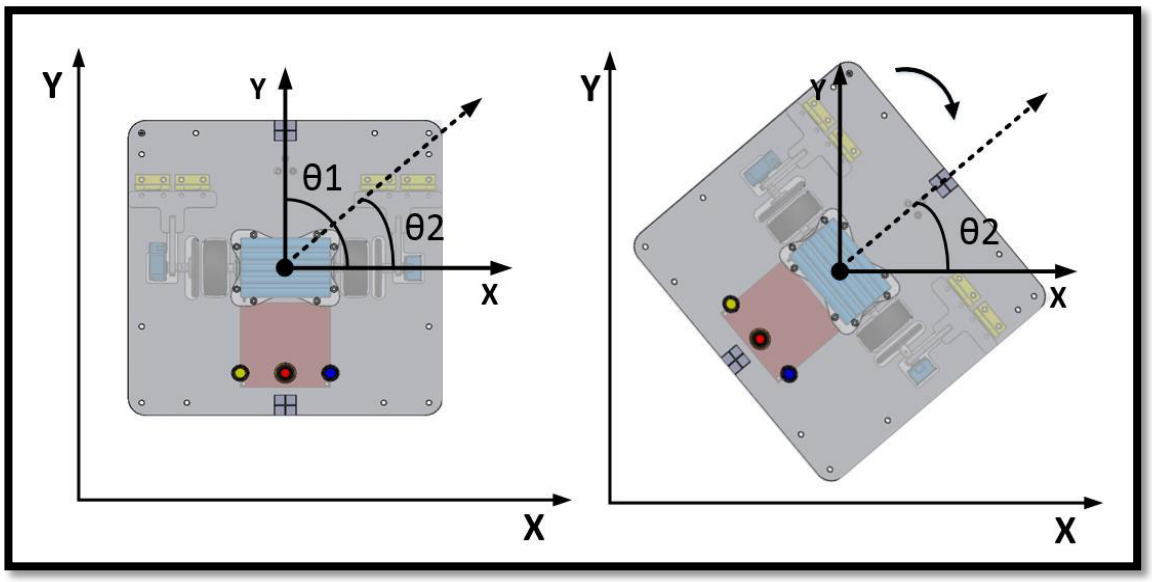


Figura 3.21.- Control angular.

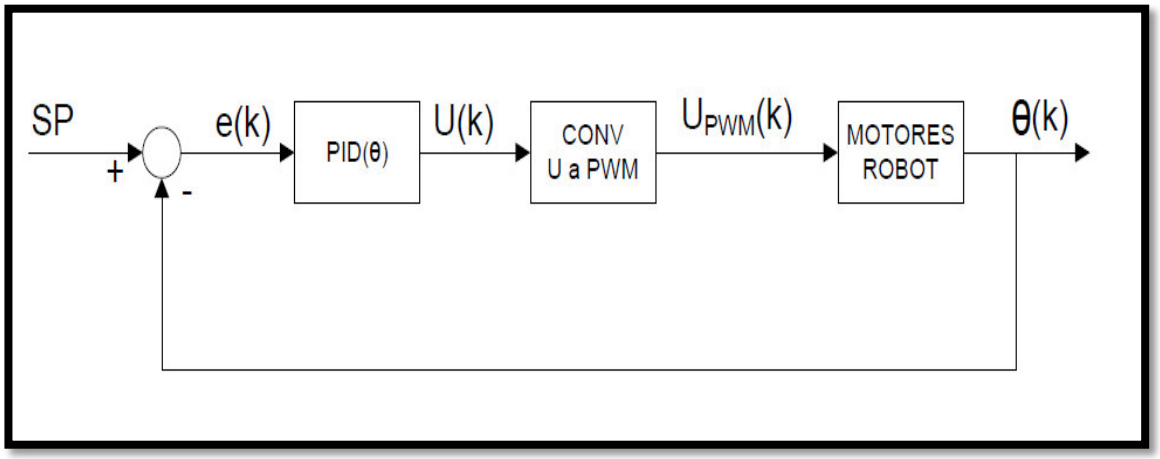


Figura 3.22.- Diagrama de bloques del sistema de control angular.

Para controlar la orientación del robot haciendo uso del diagrama de bloques (figura 3.22) se creó la función "control_ang_rot", el cual tiene como parámetros de entrada la referencia de ángulo (angulo_ref) y la histéresis. El algoritmo de esta función se muestra en la figura 3.23.

El algoritmo empieza inicializando la variable "angulo_sens" con el valor de la variable global "angulo_actual". Se calcula el error "error_angulo" como la diferencia entre la variable de entrada "angulo_ref" y "angulo_sens". Para que nuestro sistema de control sea eficiente se desarrolló una lógica para que el robot realice el menor desplazamiento angular al momento de controlar el ángulo, debido a que existen dos sentidos de giro (horario y anti-horario) para llegar al ángulo de referencia o SP. La idea es que el robot gire en el sentido que represente un menor recorrido. El primer paso es encontrar el error absoluto "error_abs" calculando el valor absoluto de "error_angulo". El segundo paso es preguntar si la variable "flag_modelo_ang" (flag de selección de tipo de optimización) es igual a cero y si "error_abs" es mayor a pi (significa que el recorrido en el otro sentido es menor) y si "error_angulo" es menor a cero (significa que "angulo_sens" es mayor a "angulo_ref") entonces le asignamos a "flag_modelo_ang" el valor de 1. Caso contrario debemos preguntar si la variable "flag_modelo_ang" es igual a cero y si "error_abs" es mayor a pi y si "error_angulo" es mayor a cero (significa que "angulo_sens" es menor a "angulo_ref") entonces le asignamos a "flag_modelo_ang" el valor de 2. El tercer paso es preguntar si "flag_modelo_ang" es igual a 1 (primer caso, donde "angulo_sens" es mayor a "angulo_ref") y si "angulo_sens" es mayor a pi, entonces se resta 2pi a "error_angulo" para que gire en el otro sentido recorriendo una distancia angular. Caso contrario se preguntará si "flag_modelo_ang" es igual a 2 (segundo caso, donde "angulo_sens" es menor a "angulo_ref") y si "angulo_sens" es menor a pi, entonces se suma 2pi a "error_angulo". Con estos tres pasos se logra hacer la optimización en el sentido de giro del robot al momento de controlar la orientación. Luego se hace mención a la histéresis, el cual es un error mínimo aceptado, por lo que se emplea la ecuación 3.11, haciendo la pregunta si "error_angulo" es mayor al valor absoluto de "histéresis" (ecuación 3.10). Si se cumple esta condición significa que aún el robot no llega al ángulo deseado por lo que se ejecutará el pid_angular y la función retornará el valor 0 como dato de que aún no se llega al estado estacionario. Caso contrario, significa que ya se llegó al SP deseado, entonces se inicializa la variable "flag_modelo_ang" y la función retornará el valor 1 como valor de haber llegado al estado estacionario.

$$error\_angulo > |\, histeresis \,| \quad \ldots (3.3)$$

$$\boldsymbol{error\_angulo} > histeresis \quad \vee \quad \boldsymbol{error\_angulo} < -1 \times histeresis \quad \ldots (\mathbf{3.4})$$

control_ang_rot(angulo_ref, histeresis)

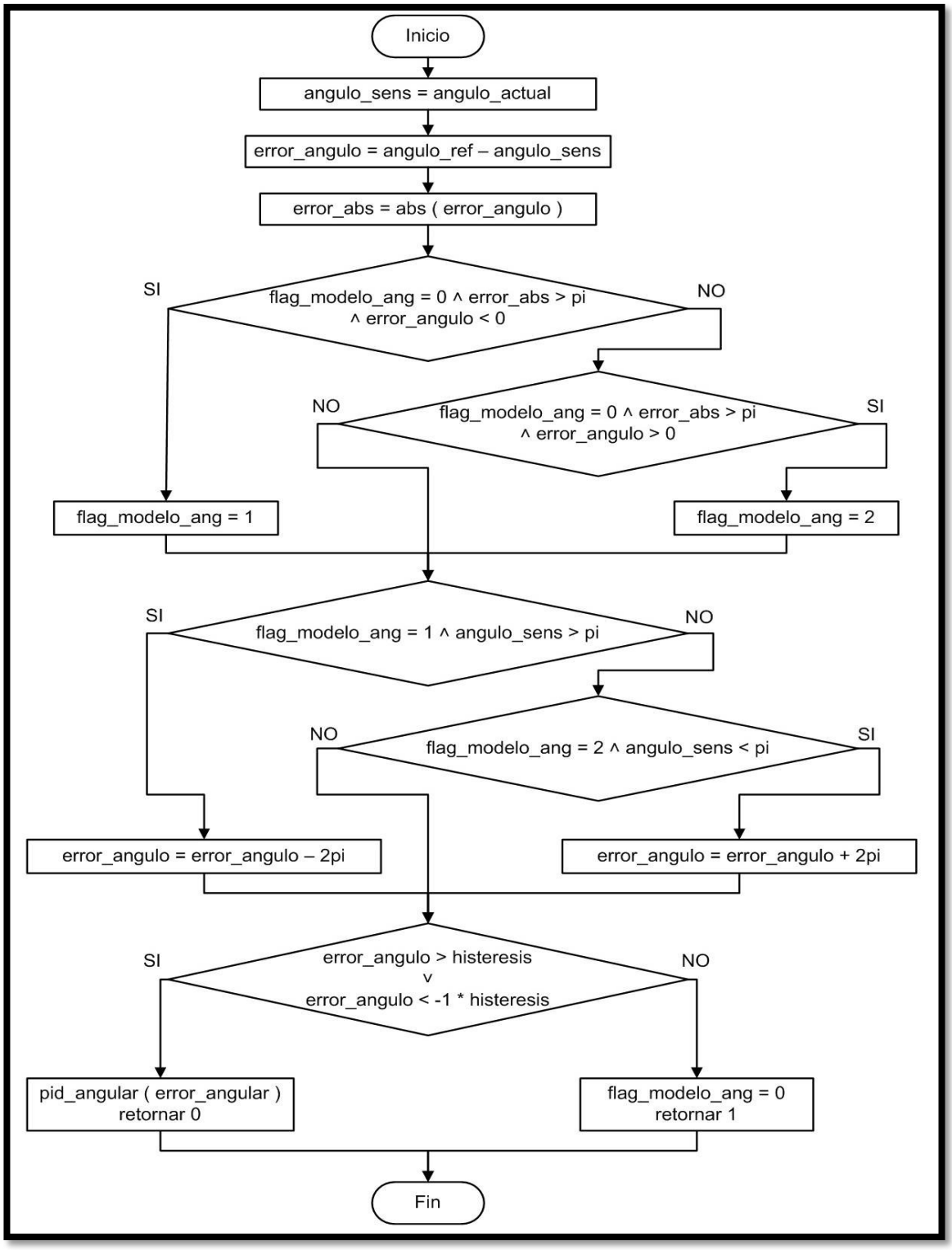


Figura 3.23.- Diagrama de flujo del control de orientación.

De acuerdo al algoritmo de control angular se tiene que implementar un controlador discreto tomando como referencia las ecuaciones del Capítulo 1, 4.Controlador PID. En el desarrollo del algoritmo del controlador PID discreto se empleará la ecuación 1.32. la cual desarrolla la ecuación en diferencias de las tres componentes: proporcional, integral y derivativa.

Lo más importante al momento de implementar un controlador PID son las constantes de sintonización Kp, Ki y Kd. Debido a que dichas constantes definirán el comportamiento de la señal de control hacia los motores. Dichas constantes fueron halladas tras una serie de pruebas dinámicas con el robot ya implementado. Estas pruebas consistieron en una toma de datos en tiempo real sobre una superficie lisa libre de imperfecciones. Estas constantes son las siguientes.

$$\boldsymbol{Kp = 1000}$$
$$\boldsymbol{Ki = 0.01}$$
$$\boldsymbol{Kd = 13500}$$

Cabe resaltar que las constantes Ki y Kd ya cuentan con el factor de tiempo de muestreo T, cuyo valor es **2.5ms**.

La implementación del controlador PID se realizó a través de la función "pid_angular", la cual tiene como parámetro de entrada el error angular "error_angulo". El diagrama de flujo de esta función se muestra en la figura 3.24. El algoritmo empieza inicializando las constantes de sintonización con los valores ya mencionados. Luego se calcula la componente proporcional "P_angulo" como la multiplicación de "Kp" con "error_angulo". También se halla la constante integral "I_angulo", la cual posee un término de acumulación "I_angulo_anterior". Además se calcula la constante diferencial "D_angulo" como la multiplicación de "Kd" con la diferencia entre "error_angulo" y "error_angulo_anterior". Finalmente se suman las tres componentes para hallar la señal de control "U_angulo". Esta variable es ingresada a la función motores para la conversión a señal PWM. El algoritmo finaliza igualando "error_angulo_anterior" a "error_angulo" y "I_angulo_anterior" a "I_angulo" para mantener la relación en la ecuación recursiva.

pid_angular(error_angulo)

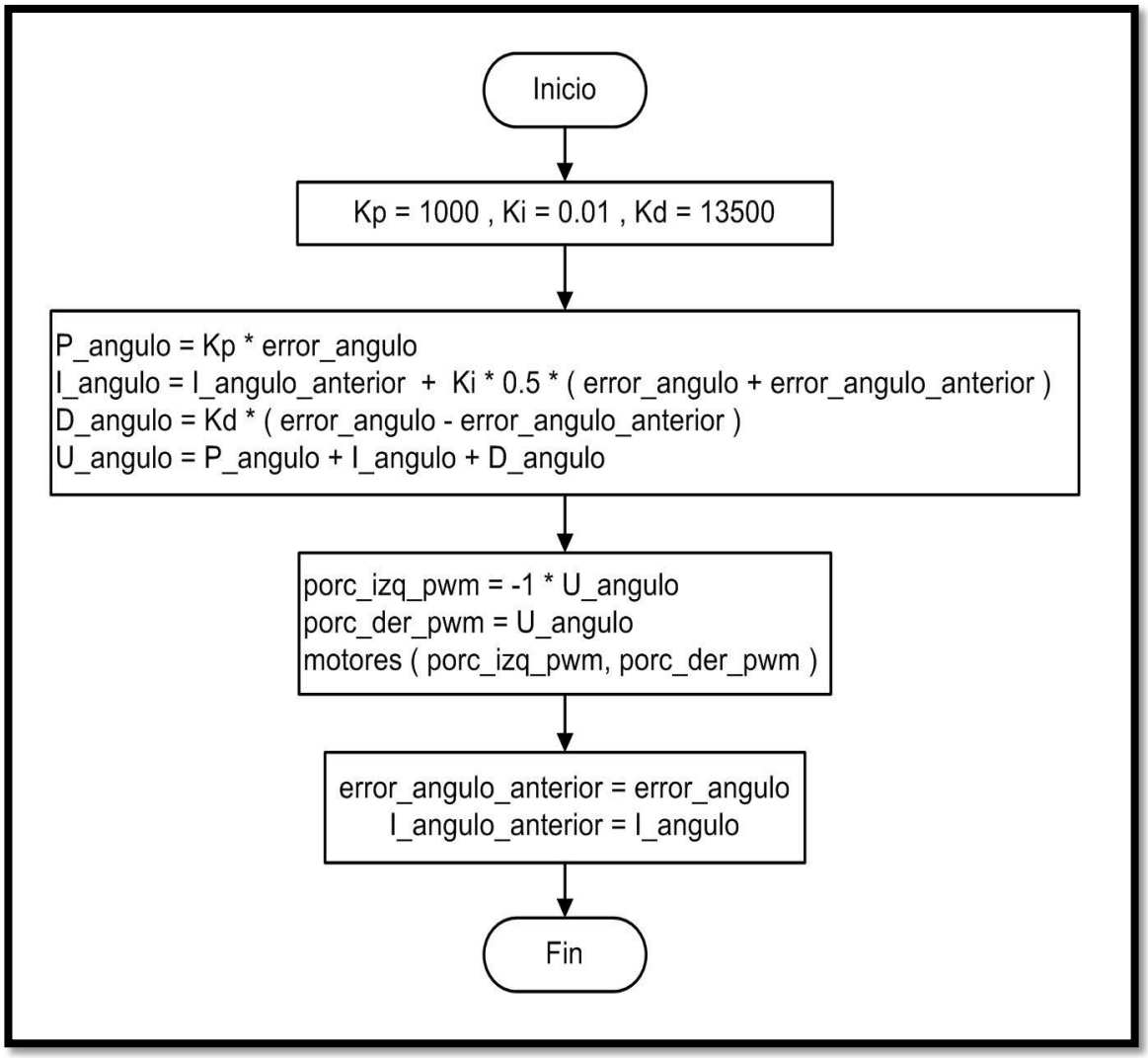


Figura 3.24.- Diagrama de flujo del controlador PID discreto de orientación.

La implementación de los algoritmos de control angular desarrollados en lenguaje C se muestran a continuación.

Primero se declaran e inicializan las variables globales en las líneas 1, 2, 3, 4, 5, 6 y 7. Estas variables serán usadas en las funciones internas del control angular como en funciones de otras librerías.

```
1.    int flag_modelo_ang = 0;
2.    float error_angulo_anterior = 0;
3.    float P_angulo = 0;
```

```
4.      float I_angulo = 0;
5.      float I_angulo_anterior = 0;
6.      float D_angulo = 0;
7.      float U_angulo = 0;
```

Luego se definen las funciones "control_ang_rot" para el control de la orientación y la función del controlador PID discreto "pid_angular".

```
8.      int control_ang_rot(float angulo_ref, float histeresis);
9.      void pid_angular(float error_angulo);
```

La implementación del diagrama de flujo del algoritmo de la figura 3.19 se muestra a partir de la línea 10. En las líneas 20 y 24 se observan las funciones de retorno de datos cuando el sistema está en estado transitorio o estacionario respectivamente.

```
10.     int control_ang_rot(float angulo_ref, float histeresis)
        {
11.         float angulo_sens = angulo_actual;
12.         float error_angulo = angulo_ref - angulo_sens;
13.         float error_abs=fabs(error_angulo);

14.         if( flag_modelo_ang == 0 && error_abs > pi && error_angulo < 0.0 ) flag_modelo_ang = 1;
15.     else if( flag_modelo_ang == 0 && error_abs > pi && error_angulo > 0.0) flag_modelo_ang = 2;

16.         if( flag_modelo_ang == 1 && angulo_sens > pi ) error_angulo += 2.0*pi;
17.     else if( flag_modelo_ang == 2 && angulo_sens < pi )  error_angulo -= 2.0*pi;

18.         if( error_angulo > histeresis || error_angulo < (-1.0*histeresis) )       {
19.             pid_angular( error_angulo );
20.             return 0;
            }
21.         else
            {
22.             flag_modelo_ang=0;
23.             I_angulo_anterior=0;
24.             return 1;
```

```
    }
}
```

Finalmente se implementa el controlador PID discreto a partir de la línea 25.

```
25.  void pid_angular(float error_angulo)
     {
26.      float Kp = 1000.0, Ki = 0.01, Kd = 13500.0;

27.      P_angulo = Kp*error_angulo;
28.      I_angulo = I_angulo_anterior + Ki*0.5*(error_angulo + error_angulo_anterior);
29.      D_angulo = Kd * (error_angulo - error_angulo_anterior);
30.      U_angulo = P_angulo + I_angulo + D_angulo;

31.      porc_izq_pwm = -1.0 * U_angulo;
32.      porc_der_pwm = U_angulo;
33.      motores(porc_izq_pwm,porc_der_pwm);

34.      error_angulo_anterior = error_angulo;
35.      I_angulo_anterior=I_angulo;
     }
```

3.3 Control de posición

El control de posición regulará el desplazamiento lineal del robot cuando se dirija de un punto a otro. En la figura 3.25 se observa que el robot se encuentra en el punto P1 con una orientación θ, se requiere que se desplace hasta el punto P2 con la misma orientación. El móvil deberá recorrer una distancia L en línea recta para llegar al punto deseado. Se utilizó un controlador PID de posición para manipular la velocidad de los motores con el fin que el robot llegue al punto deseado con un mínimo error aceptable.

El sistema de control de posición está representado en el diagrama de bloques mostrado en la figura 3.26. Al igual que en el sistema de control anterior se cuenta con un SP o referencia de posición. El SP se comparará con la salida del sistema, dando como resultado un error (e). El error ingresará como variable de entrada al controlador

PID. El resultado del algoritmo del PID será la señal de control U, la cual se transformará a una salida de PWM para luego ser transmitidas hacia los motores, los cuales generarán una variación en la posición del robot.

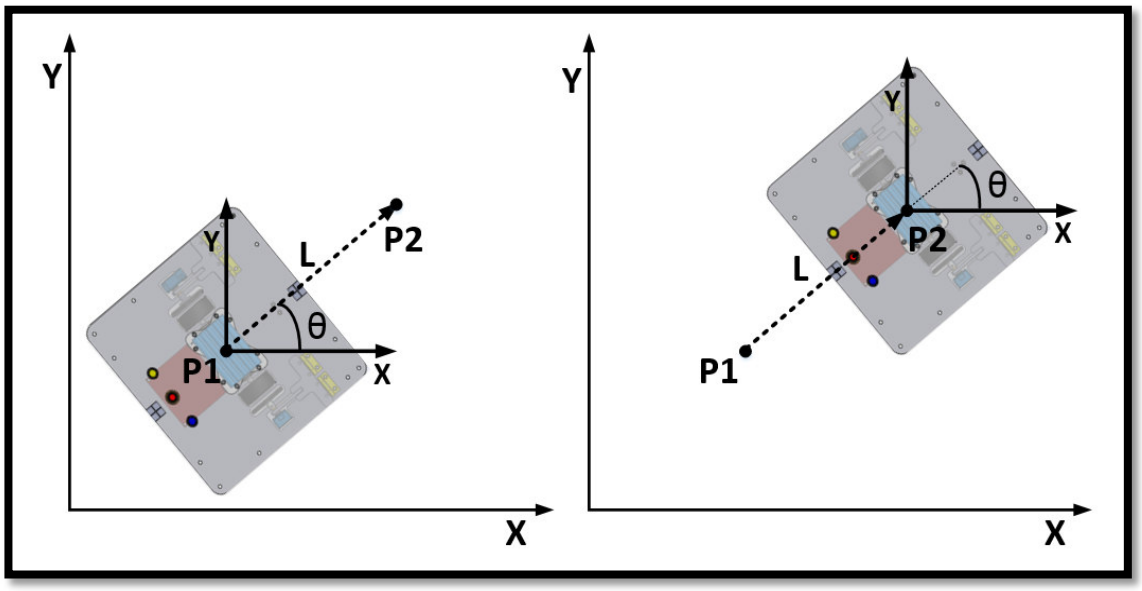


Figura 3.25.- Control de posición.

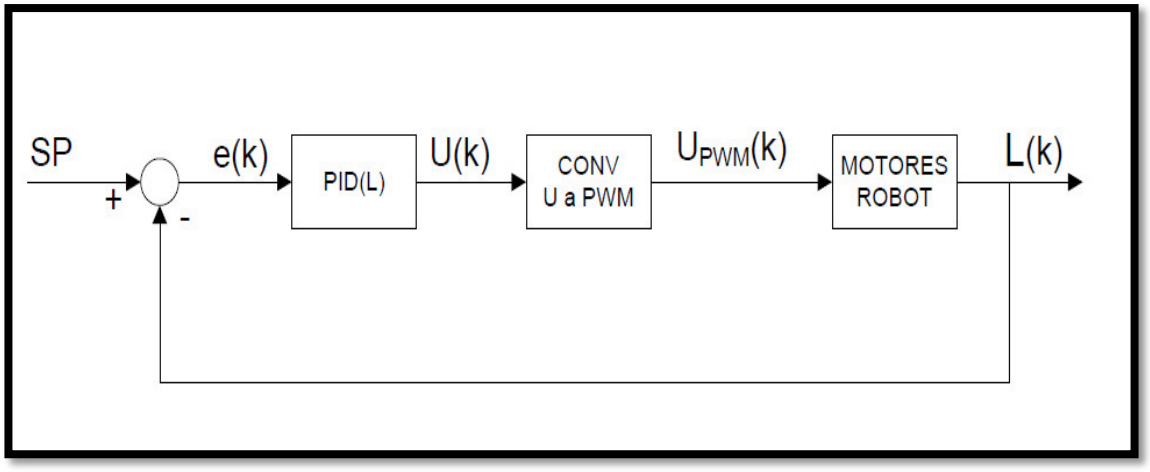


Figura 3.26.- Diagrama de bloques del control de posición.

Para controlar la posición del robot teniendo como referencia el diagrama de bloques de control, se creó la función "control_posicion", la cual tiene como parámetros de entrada las coordenadas "X" e "Y" así como la histéresis. El algoritmo de esta función se observa en la figura 3.27. Se inicializa preguntando si la variable global "flag_pos" es igual a 0, quiere decir si es la primera vez que se ejecuta esta función. Si es afirmativa la condición se procede a igualar "X_ini" y "Y_ini" a las variable globales de la librería de odometría "X_actual" y "Y_actual" respectivamente. Además se calcula la referencia o SP de posición lineal empleando la función "dist_ptos". Luego se iguala la variable "flag_pos" a 1 para que la siguiente vez que se ejecute la función no vuelva a inicializar "X_ini" y "Y_ini" ni tampoco calcule el SP nuevamente, sino que se mantenga el calculado la primera vez.

El error de posición "error_pos" se calcula como la diferencia entre el SP o referencia "L_ref" y la distancia recorrida, la cual se haya a través de la función "dist_ptos". Esta función calcula la distancia entre dos coordenadas, por lo cual se ingresan como datos de entrada "X_actual" y "Y_actual" como primer punto; "X_ref" y "Y_ref" como segundo punto. Finalmente se pregunta si el error es mayor al valor absoluto de "histeresis". Si es mayor, lo cual significa que aún no se llega al punto deseado, se ejecutará el algoritmo de control PID discreto y se retornará el valor de 0 para indicar que aún no se llega al estado estacionario sino que sigue en el transitorio. Caso contrario se detendrá el robot a través de la función "motores" con parámetros 0 y se retornará el valor de 1 indicando que ya se llegó al estado estacionario.

control_posicion(X_ref, Y_ref, histéresis)

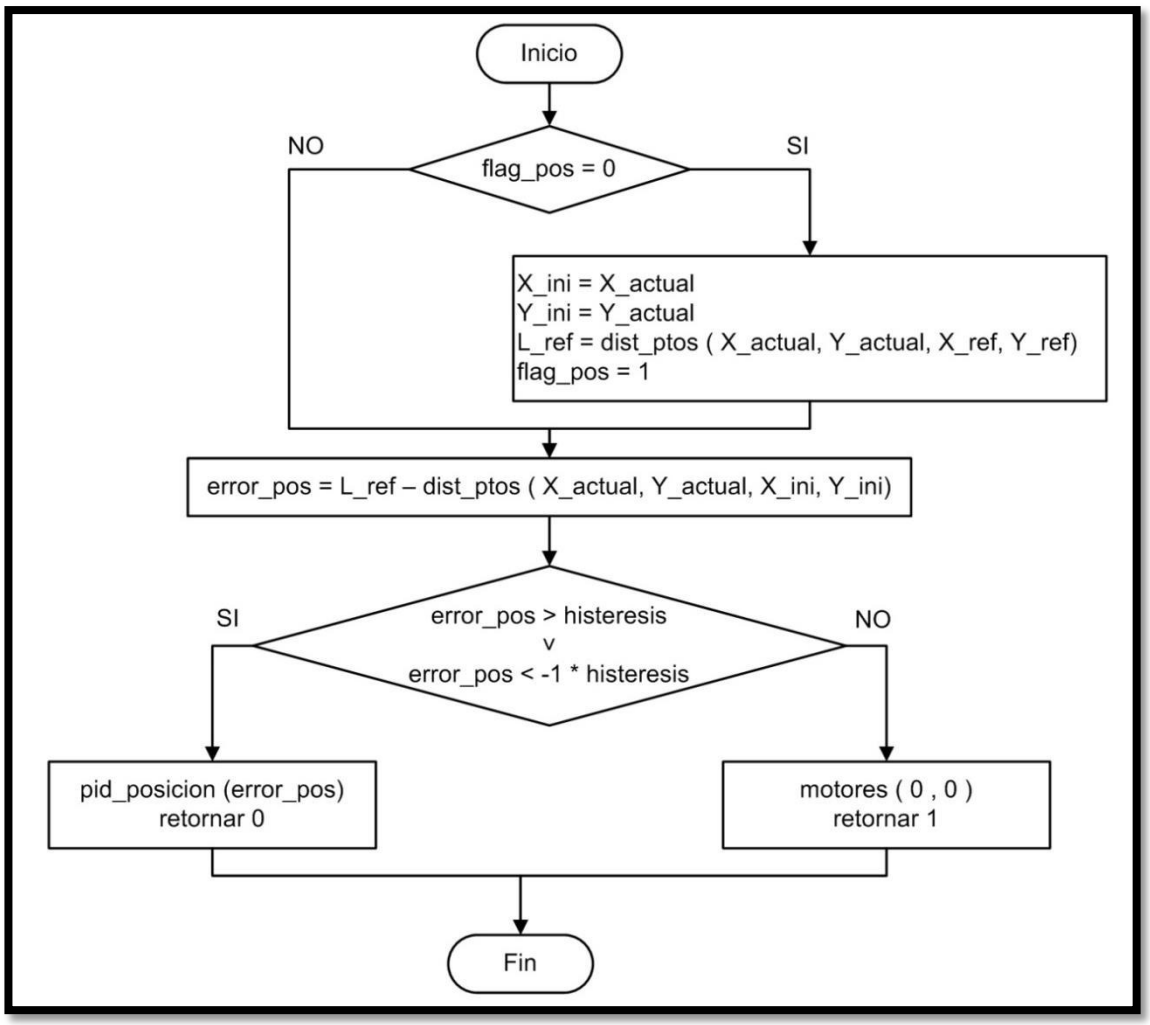


Figura 3.27.- Diagrama de flujo del control de posición.

Al igual que en el algoritmo de control de posición se tendrá que implementar un controlador PID discreto de posición de manera similar al PID angular. En este caso también se hallaron las constantes tras una serie de pruebas dinámicas en el robot sobre una superficie milimetrada libre de imperfecciones. Los resultados obtenidos al finalizar las pruebas dieron a conocer las constantes indicadas para el control de posición del robot. Estas constantes son las siguientes.

$$\boldsymbol{Kp = 50} \qquad \boldsymbol{Ki = 0.001} \qquad \boldsymbol{Kd = 800}$$

Cabe resaltar que las constantes Ki y Kd ya cuentan con el factor de tiempo de muestreo T, cuyo valor es 2.5 ms.

La implementación de este controlador PID se realizó a través de la función "pid_posicion", la cual tiene como parámetro de entrada al error de posición "error_pos". El algoritmo de esta función se puede observar en la figura 3.28, el cual empieza inicializando las constantes de sintonización a los valores hallados en las pruebas dinámicas. Luego se calcula la componente proporcional "P_pos" como la multiplicación de "Kp" con "error_pos". También se halla la constante integral "I_pos", la cual posee un término de acumulación "I_pos_anterior". Además se calcula la constante diferencial "D_pos" como la multiplicación de "Kd" con la diferencia entre "error_pos" y "error_pos_anterior". Finalmente se suman las tres componentes para hallar la señal de control "U_pos". Esta variable es asignada a la función motores para que se realice la conversión a señal de PWM. Finalmente se iguala "error_pos_anterior" a "error_pos" y "I_pos_anterior" a "I_pos" para mantener la relación en la ecuación recursiva.

pid_posicion(error_pos)

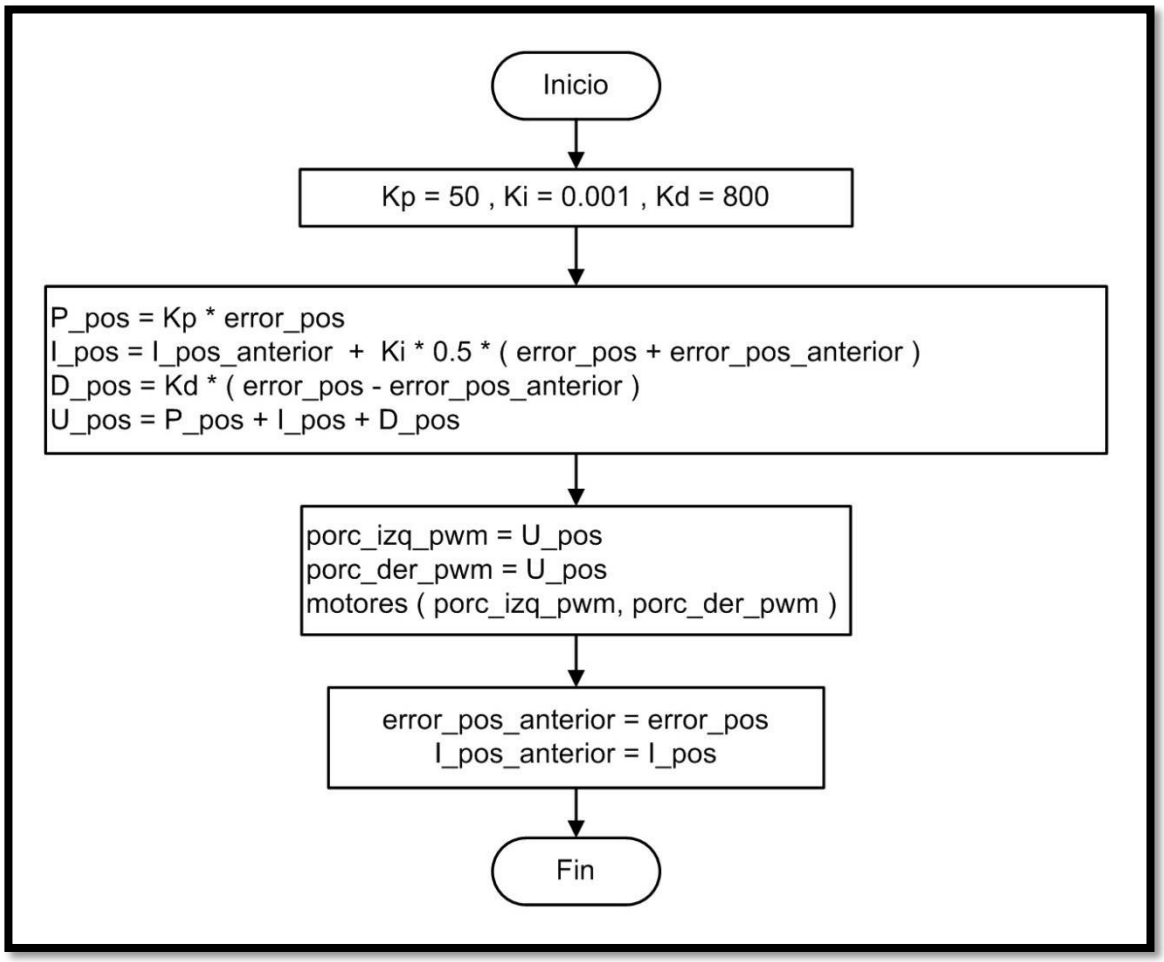


Figura 3.28.- Diagrama de flujo del controlador PID de posición.

De acuerdo al algoritmo de control de posición se tiene que implementar una función que calcule la distancia entre dos puntos [19]. Este cálculo se desarrolla a partir de la ecuación 3.4.

$$d = \sqrt{(x1 - x2)^2 + (y1 - y2)^2} \ldots (3.4)$$

El diagrama de flujo de esta función se muestra en la figura 3.29.

dist_ptos(x1, y1, x2, y2)

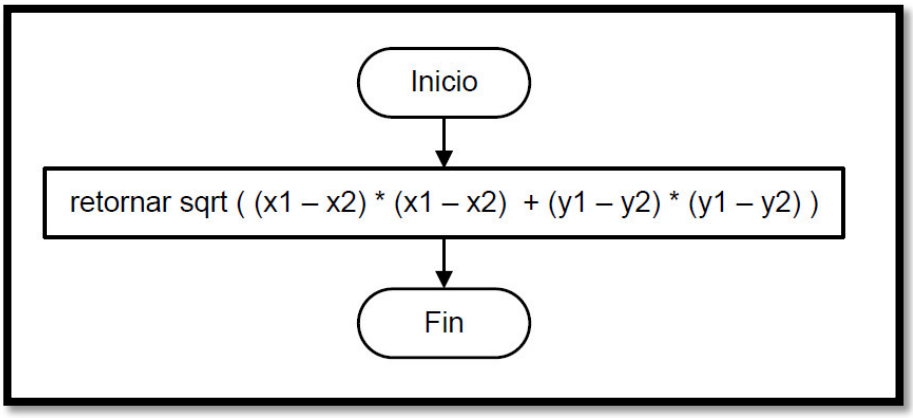


Figura 3.29.- Diagrama de flujo de la función "dist_ptos".

La implementación de los algoritmos referentes al control de posición se desarrolló en lenguaje C de la siguiente manera.

Primero se declaran e inicializan las variables globales desde la línea 1 hasta la 10. Estas variables serán usadas en las funciones internas del control de posición como en funciones de otras librerías.

```
1.  int flag_pos = 0;

2.  float X_ini = 0;
3.  float Y_ini = 0;
4.  float L_ref = 0;

5.  float error_pos_anterior = 0;
6.  float P_pos = 0;
7.  float I_pos = 0;
```

```
8.      float I_pos_anterior = 0;
9.      float D_pos = 0;
10.     float U_pos = 0;
```

Luego se definen las funciones "control_posicion" para el control de la posición; "pid_posicion" para el controlador PID; y "dist_ptos" para calcular la distancia entre dos puntos o coordenadas.

```
11.     int control_posicion ( float X_ref,  float Y_ref,  float hysteresis );
12.     void pid_posicion ( float error_pos );
13.     float dist_ptos ( float x1, float y1, float x2, float y2 );
```

La implementación del diagrama de flujo del algoritmo de la figura 3.27 se muestra a partir de la línea 14. En las líneas 24 y 30 se observan las funciones de retorno de datos cuando el sistema está en estado transitorio y estacionario respectivamente. Además en la línea 19 se hace uso de la función "dist_ptos" para calcular la distancia que el robot deberá recorrer entre la posición actual ("X_actual" y "Y_actual") y la posición deseada o de referencia ("X_ref" y "Y_ref").

```
14.     int control_posicion ( float X_ref,  float Y_ref,  float hysteresis )
        {
15.         float error_pos;
16.         if( flag_pos == 0 )
            {
17.             X_ini = X_actual;
18.             Y_ini = Y_actual;
19.             L_ref = dist_ptos( X_actual, Y_actual, X_ref, Y_ref);
20.             flag_pos = 1;
        }

21.     error_pos = L_ref - dist_ptos( X_actual, Y_actual, X_ini, Y_ini );
22.     if ( error_pos > histeresis || error_pos < (-1.0*histeresis)  )
        {
23.         pid_posicion( error_pos );
24.         return 0;
        }
25.     else
```

```
        {
26.         porc_izq_pwm = 0;
27.         porc_der_pwm = 0;
28.         motores ( porc_izq_pwm, porc_der_pwm );
29.         I_pos_anterior = 0;
30.         return 1;
        }
        }
```

Ahora se implementará el controlador PID discreto de posición a partir de la línea 31. En la línea 32 se declaran e inicializan las constantes de sintonización

```
31.     void pid_posicion( float error_pos )
        {
32.         float Kp = 50.0,  Ki = 0.001,  Kd = 800.0;
33.         P_pos = Kp * error_pos;
34.         I_pos = I_pos_anterior  +  Ki * 0.5 * ( error_pos  +  error_pos_anterior );
35.         D_pos = Kd*( error_pos  -  error_pos_anterior );
36.         U_pos = P_pos + I_pos + D_pos;

37.         porc_izq_pwm = U_pos;
38.         porc_der_pwm = U_pos;
39.         motores ( porc_izq_pwm, porc_der_pwm );

40.         error_pos_anterior = error_pos;
41.         I_pos_anterior = I_pos;
        }
```

Finalmente se implementa la función "dist_ptos", en la cual se retorna la distancia calculada a partir de la ecuación 3.4.

```
42.     float dist_ptos( float x1,  float y1,  float x2,  float y2 )
        {
43.         return sqrt( pow( x1 - x2, 2.0 ) +  pow( y1 - y2 ,  2.0 ) );
        }
```

3.4 Control de distancia a la recta

Este control tiene la finalidad de evitar que el robot se desvíe de la trayectoria trazada durante su recorrido. Supongamos el móvil se dirige del punto P1 al punto P2 con una orientación θ1 tal como se muestra en la figura 3.30. En el caso que hubiese una desviación en la trayectoria del móvil por algún motivo como se ve en la figura 3.31, donde la orientación del robot cambió a θ2, desviándose de la recta trazada inicialmente. En este punto el controlador PID buscará corregir el cambio de orientación como se observa en la figura 3.32. Si el robot continúa con su trayectoria no llegará al punto P2 sino al P3, el cual se encuentra en la recta paralela. Para evitar este posible error se implementará un algoritmo que consiste en calcular la distancia (d) a la recta que pasa por los puntos P1 y P2. Si la distancia supera un margen mínimo aceptado se redireccionará la trayectoria del robot hacia el punto P2, tal como se muestra en la figura 3.33.

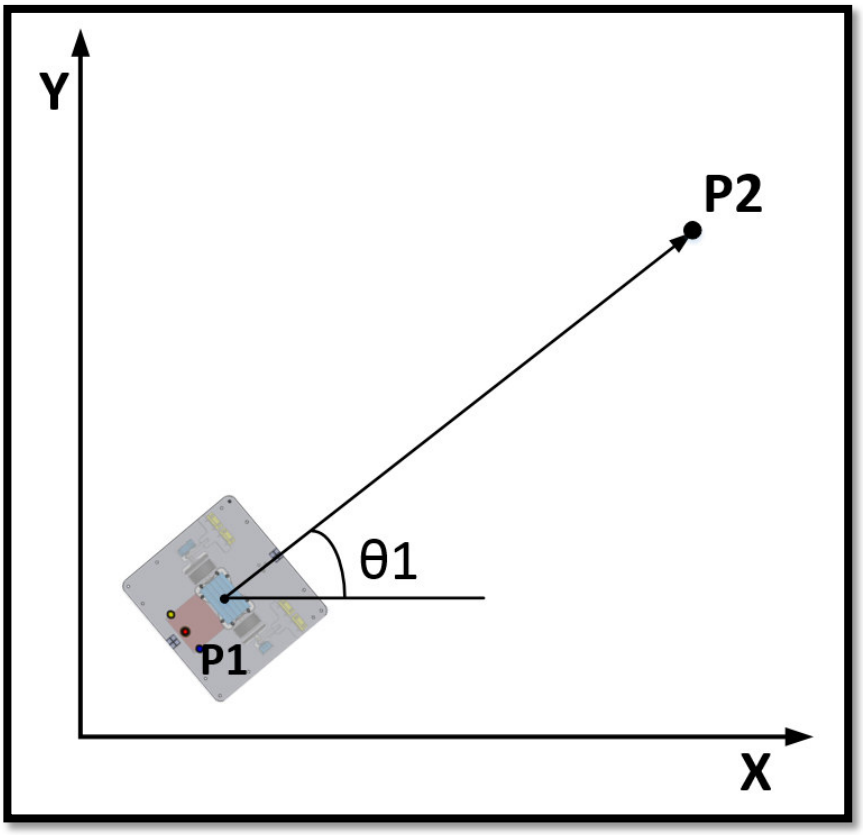


Figura 3.30.- Control de distancia a la recta - 1.

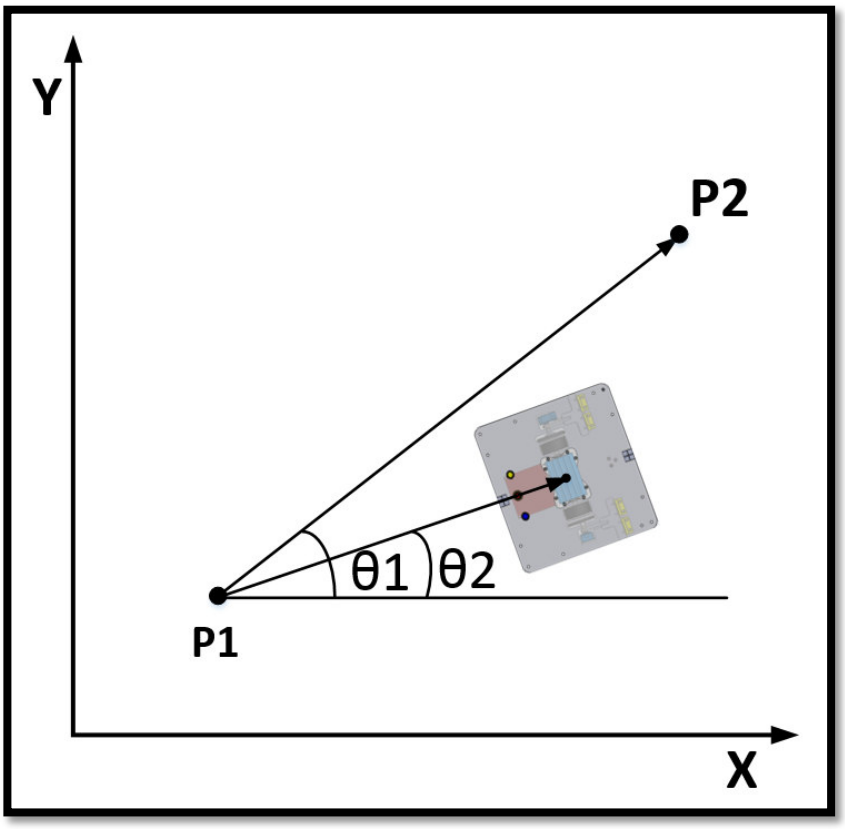


Figura 3.31.- Control de distancia a la recta - 2.

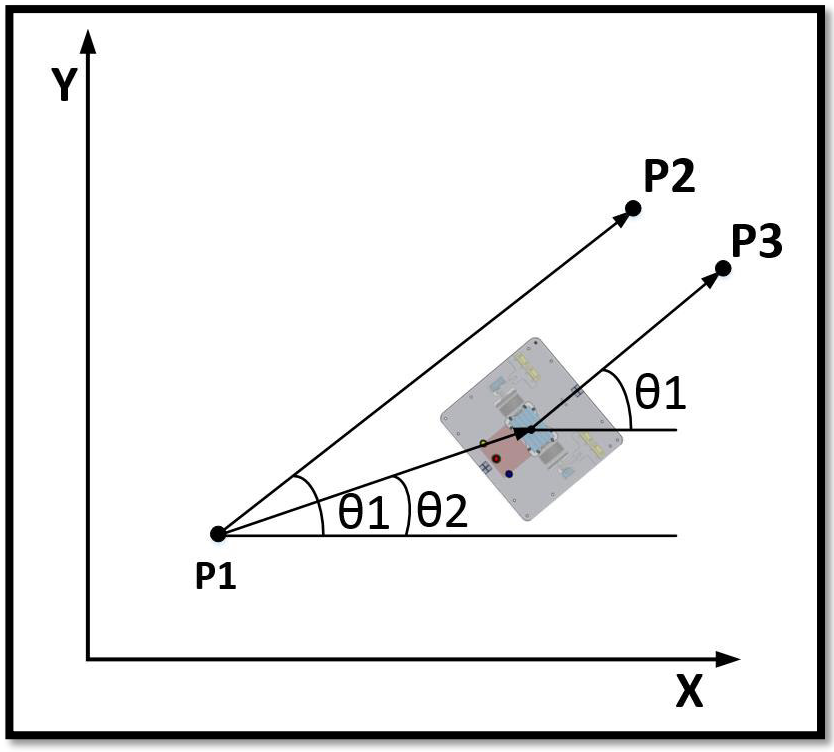


Figura 3.32.- Control de distancia a la recta – 3.

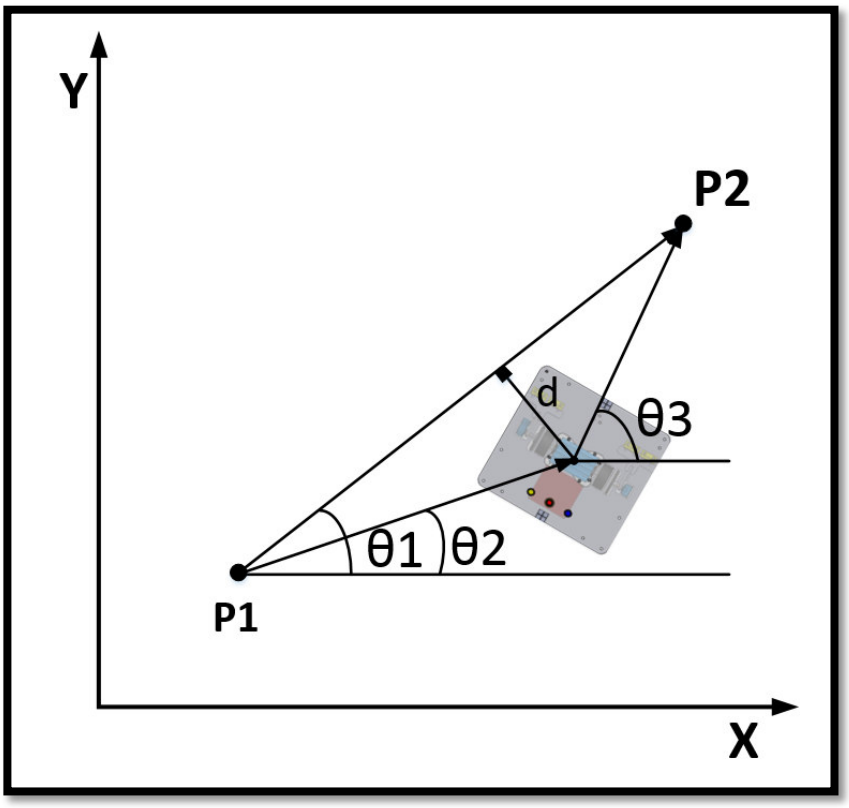


Figura 3.33.- Control de distancia a la recta – 4.

Para implementar esta lógica se creó la función "control_on_off_dist_recta", la cual tiene como parámetros de entrada los coeficientes de la recta por la cual el robot debería desplazarse para llegar de un punto a otro ("coef [3]") y una histéresis (margen mínimo aceptado). El algoritmo de esta función se muestra en la figura 3.34. Se inicia calculando la distancia a la recta la cual llamaremos "error_dist_recta" a través de la función "dist_pto_recta", la cual explicaremos luego. Una vez calculada esta variable se procederá a preguntar si es mayor a la histéresis. Si es mayor, se retornará el valor 1, indicando que se sobrepasó el límite máximo permisible de distancia a la recta. Caso contrario, se retornará el valor 0, indicando que aún no se sobrepasa el límite.

control_on_off_dist_recta (coef[3], histeresis)

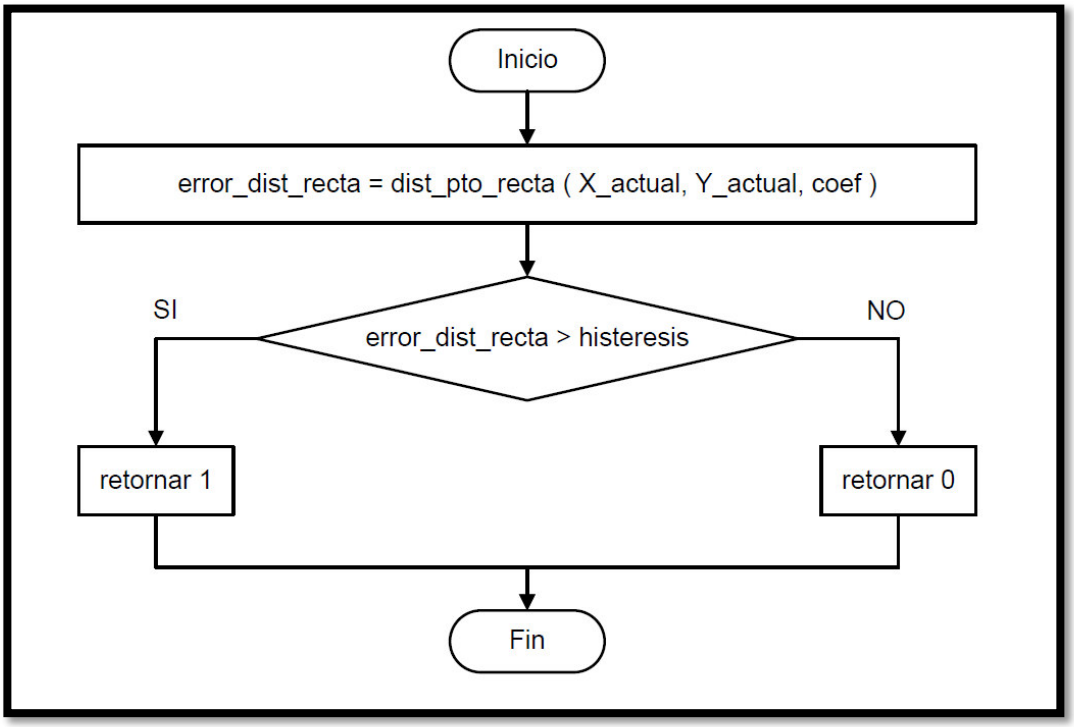


Figura 3.34.- Diagrama de flujo de la función "control_on_off_dist_recta".

Según el algoritmo anterior se requiere una función ("dist_pto_recta") para calcular la distancia a la recta desde el punto P(x,y). Esta función tiene como parámetros de entrada las coordenadas del punto P (x,y) y los coeficientes de la recta r ("coef [3]"). Para calcular la distancia se emplearán las siguientes ecuaciones [20].

Sea la recta r descrita en la ecuación 3.5

$$Ax + By + C = 0 \quad \ldots (3.5)$$

La distancia del punto P(x,y) a la recta r se calcula con la ecuación 3.6.

$$\boldsymbol{d(P,r) = \frac{|\,Ax + By + C\,|}{\sqrt{A^2 + B^2}}} \ldots \boldsymbol{(3.6)}$$

Para nuestro algoritmo la equivalencia de variables sería la siguiente.

$$A = coef[0]$$
$$B = coef[1]$$
$$C = coef[2]$$

dist_pto_recta (x, y, coef[3])

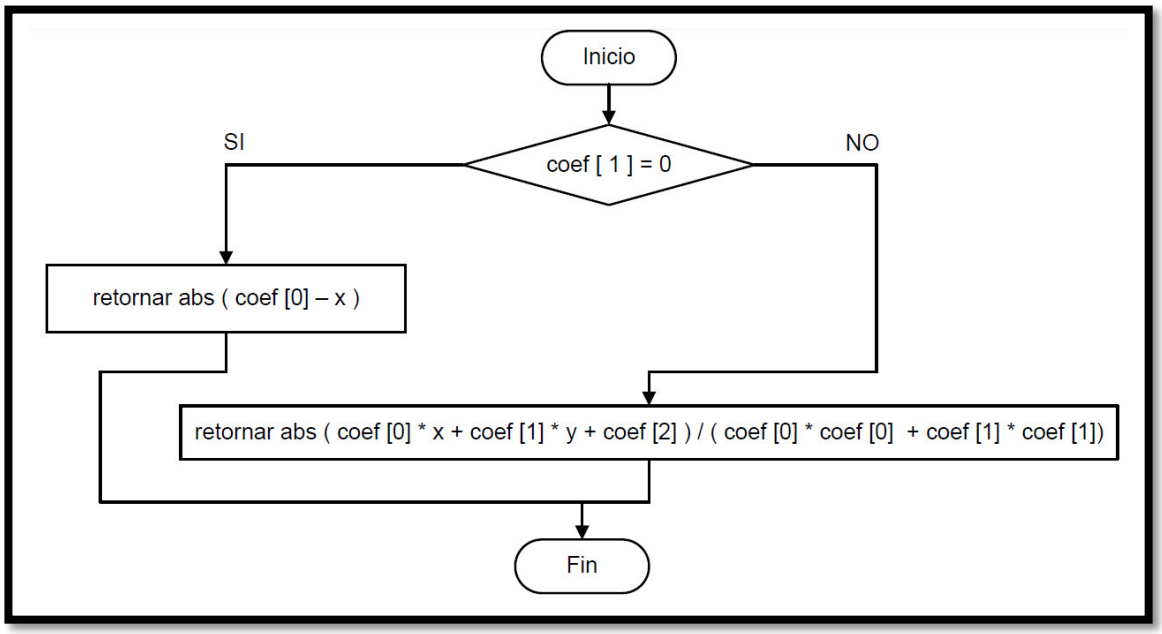


Figura 3.35.- Diagrama de flujo de la función "dist_pto_recta".

La implementación de los algoritmos referentes al control de la distancia a la recta en lenguaje C se muestran a continuación.

En la línea 1 se declara la variable global "coef_ec_recta" la cual es un vector de de tres elementos tipo real. También se declaran las funciones "control_on_off_dist_recta" y "dist_pto_recta" (líneas 2 y 3).

```
1.    float coef_ec_recta[3];

2.    int control_on_off_dist_recta ( float coef[3], float histeresis );
3.    float dist_pto_recta ( float x,  float y,  float coef[3] );
```

A partir de la línea 4 se implementa la función "control_on_off_dist_recta" del algoritmo de la figura 3.26. En la línea 5 se declara e inicializa el valor de "error_dist_recta" con el resultado de la ecuación "disp_pto_recta".

```
4.    int control_on_off_dist_recta( float coef[3], float histeresis)
      {
5.        float error_dist_recta = dist_pto_recta (X_actual, Y_actual, coef);

6.        if( error_dist_recta  > histeresis )
7.            return 1;
8.        else
9.            return 0;
      }
```

Finalmente se codifica la función "dist_pto_recta" del algoritmo de la figura 3.27.

```
10.   float dist_pto_recta(float x, float y,float coef[3])
      {
11.     if( coef[1]==0 )
12.       return fabs( coef[0] - x );
13.     else
14.       return fabs((coef[0]*x + coef[1]*y + coef[2]) / ( sqrt (pow (coef[0],2.0) + pow(coef[1],2))));
      }
```

4. Desplazamiento del robot

El robot tiene la capacidad de posicionarse en diferentes puntos de manera secuencial. Por ejemplo en la figura 3.36 se observa que el robot se dirige secuencialmente del punto P1 a los puntos P2, P3 y P4. Se realizará un control de posición (ver capítulo 3, 3.Controlador) para el controlar desplazamiento entre puntos. Así como un control angular para regular la orientación del robot al momento de cambiar de trayectoria.

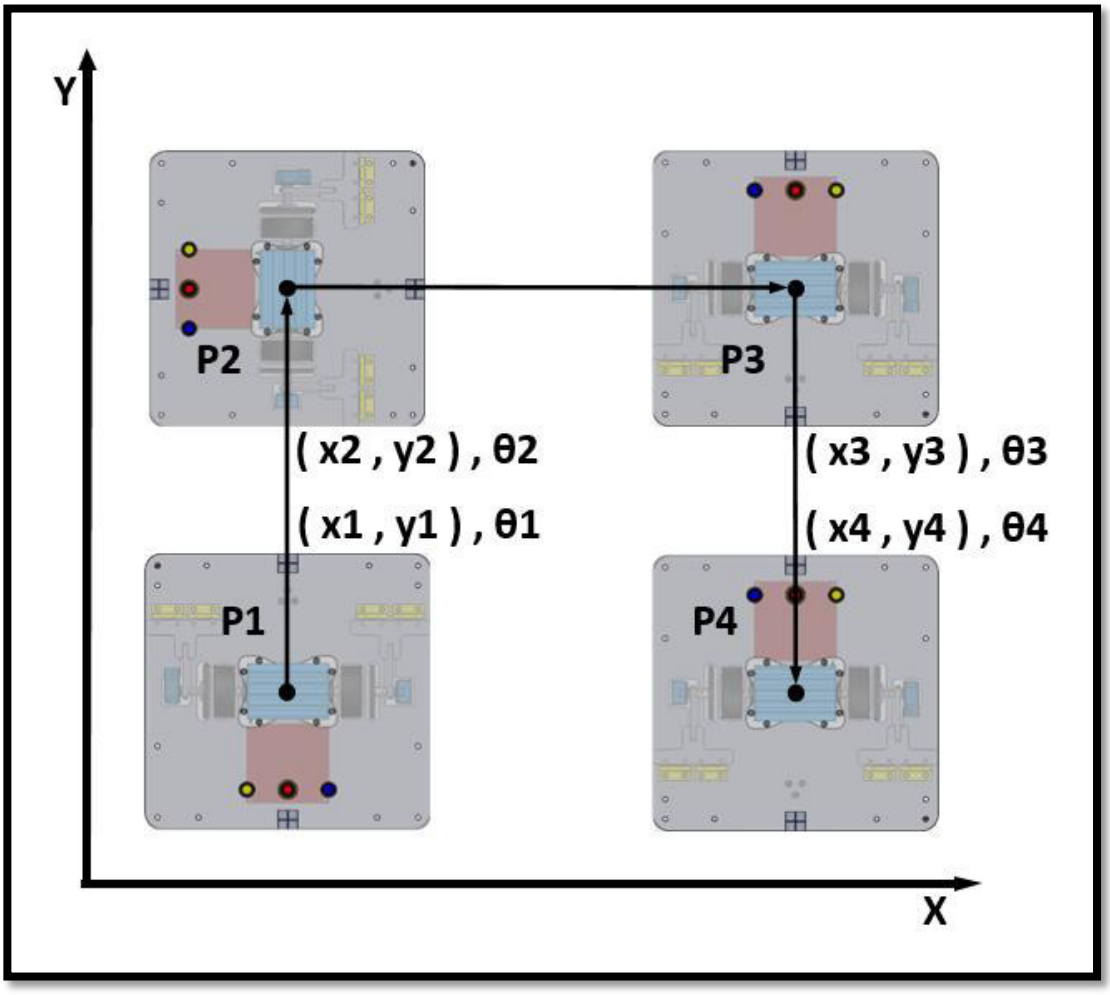


Figura 3.36.- Desplazamiento del robot entre puntos secuenciales.

Para tener una secuencia ordenada el sistema tendrá un arreglo o matriz de coordenadas de los puntos en los cuales se deberá posicionar el robot, tal como se muestra en la figura 3.37. Esta matriz posee como elementos el tipo de movimiento, las coordenadas del punto al cual debe dirigirse el robot y la orientación deseada. El

tipo de movimiento puede ser: traslación, rotación o una espera en la secuencia. Cuando se ejecuta la traslación se toman las coordenadas XY como referencia para realizar el control de posición, no se tomará en cuenta el parámetro de orientación deseada. Cuando se ejecuta la rotación, se realizará un control angular tomando como referencia el parámetro de orientación, no se tomará en cuenta las coordenadas XY. Finalmente, se puede ejecutar una espera en la secuencia en la cual el robot se mantiene detenido esperando la orden para continuar.

$$(tipo\ de\ movimiento,\quad coordenada\ X,\quad coordenada\ Y,\quad ángulo\ de\ orientación)$$

(	1,	x1,	y1,	'N'	)
(	1,	x2,	y2,	'N'	)
(	1,	x3,	y3,	'N'	)
(	1,	x4,	y4,	'N'	)
(	2,	'N',	'N',	'N'	)

Figura 3.37.- Matriz de coordenadas.

Donde:

- **Primer elemento** : Tipo de movimiento
  - 0 : rotación
  - 1 : traslación
  - 2 : espera
- **Segundo elemento**: coordenada X
- **Tercer elemento**: coordenada Y
- **Cuarto elemento**: ángulo de orientación.

La ejecución de cada fila de la matriz será de forma secuencial y continuará con la siguiente una vez que el robot termine de realizar dicho movimiento.

Con este arreglo se puede contar con una serie de puntos en los cuales el robot se posicionará secuencialmente. Además, el sistema cuenta con la funcionalidad de agregar filas a la matriz en línea, quiere decir que se pueden agregar mientras el robot está ejecutando la secuencia. Este algoritmo fue implementado creando la función "add_coordenadas" (ver figura 3.38. Dicho comando puede ser ejecutado mediante una orden enviada desde la computadora, lo que permite que se puedan agregar nuevas filas mientras el robot está en funcionamiento.

```
add_coordenadas(1    ,100.0  ,100.0  ,'N');
add_coordenadas(0    ,'N'    ,'N'    ,pi/2.0);
add_coordenadas(0    ,'N'    ,'N'    ,pi/2.0);
add_coordenadas(1    ,100.0  ,100.0  ,'N');
add_coordenadas(0    ,'N'    ,'N'    ,0.0);
```

Figura 3.38.- Función "add_coordenadas".

El algoritmo de la función "add_coordenadas" se muestra en la figura 3.39. Esta función tiene como dato de entrada el tipo de movimiento "tipo", la referencia de la coordenada X "X_ref", la referencia de la coordenada Y "Y_ref" y la referencia del ángulo de orientación "A_ref". Además se cuenta con una variable global inicializada en 0 llamada "indice_rutina", la cual indica la fila que se está agregando. El funcionamiento del algoritmo consiste en aumentar en 1 la variable global "indice_rutina" cada vez que se ejecute la función. Luego agregarán las variables de entrada de la función. El primer elemento será la variable "tipo", el segundo "X_ref", el tercero "Y_ref" y el cuarto "A_ref".

add_coordenadas(tipo, X_ref, Y_ref, A_ref)

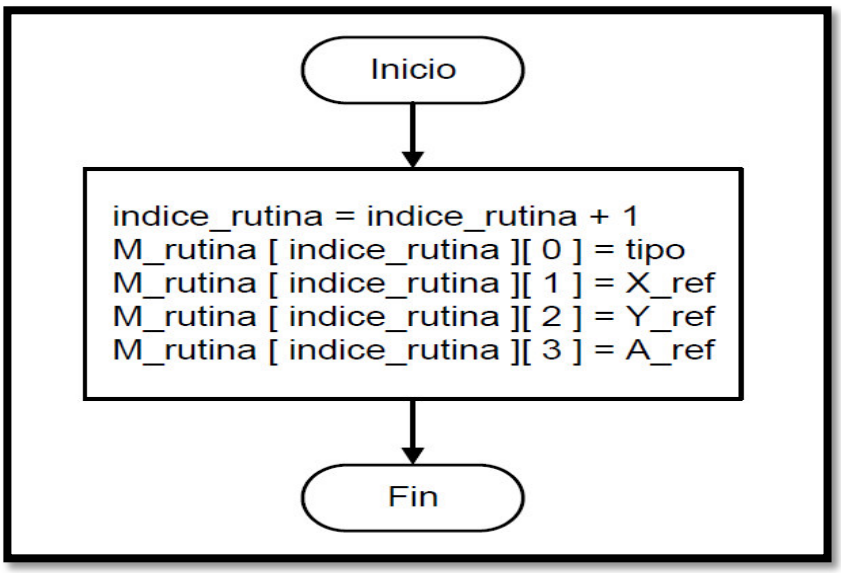


Figura 3.39.- Diagrama de flujo de la función "add_coordenadas".

El sistema administra cuando debe utilizar uno de los tipos de control, ya que ambos tienen efecto sobre los mismos motores. La prioridad de control es la siguiente:

1. Control de distancia a la recta.
2. Control angular
3. Control de posición

En la figura 3.40 se observa un diagrama de flujo simple que esquematiza la secuencia de prioridades de control. Primero se calcula la ecuación de la recta entre el punto de posición actual y el punto al que se desea llegar. Luego se ejecuta el control de distancia a la recta, si la distancia es mayor al margen o histéresis se vuelve a calcular la ecuación, caso contrario se pregunta si hay error de ángulo. Si hay error se ejecuta el control angular, sino se pregunta si hay error de posición. Si lo hubiera se controlará la posición. Esta serie de condicionales se ejecutarán en cada ciclo, por lo que siempre se tendrá dicha prioridad de control.

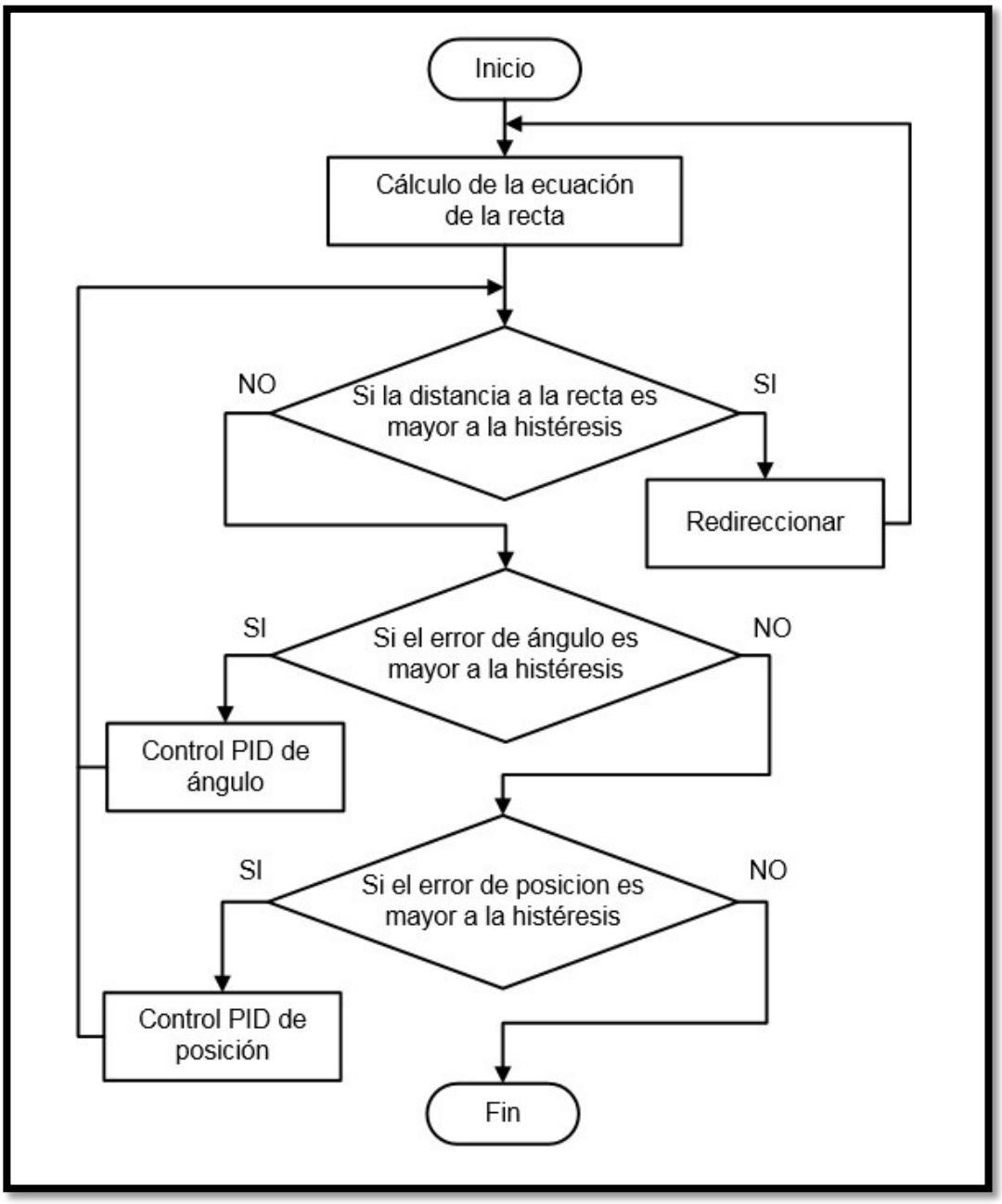


Figura 3.40.- Diagrama de prioridades de control.

El algoritmo de prioridades de control se implementó en la función "control_trayectoria" (figura 3.41), la cual tiene como parámetros de entrada el tipo de movimiento "tipo", la referencia de la coordenada X "X_ref", de la coordenada Y "Y_ref" y de la orientación angular "A_ref". El algoritmo inicia preguntando el tipo de movimiento, si es de tipo 0 (rotación) se ejecuta el control angular a través de la función "control_ang_rot", ingresando como dato de entrada el ángulo de referencia "A_ref" y una histéresis de 0.01745 radianes o 1º. Si el valor de retorno de esta función

es 1, significa que el robot llegó a la orientación deseada, por lo que se igualará la variable "flag_control" a ‘T’ (carácter que indica que se terminó la rutina de control). Si el valor es diferente a 1, quiere decir que aún se sigue ejecutando el control angular, por lo que se igualará la variable "flag_control" a 0.

Si el tipo de movimiento es igual a 1 (traslación) se empleará la secuencia de prioridad de control mostrada en la figura 3.40. La variable "flag_control" indicará que tipo control se ejecutará en función a las prioridades ya mencionadas.

Si "flag_control" es igual a 0, se calculará la ecuación de la recta que se empleará para el control de distancia a la recta a través de la función "ec_recta". También se calcula el ángulo que deberá rotar el robot ("An_ref") para orientarse en dirección al punto al que se desplazará y se iguala la variable "flag_angulo" a 1 para que se ejecute el siguiente control.

Si "flag_control" es igual a 1, se ejecutará el control de distancia a la recta. Si el valor de retorno de la función "control_on_off_dist_recta" es 1, significa que el robot se ha desviado una distancia mayor a la histéresis de 1cm y se igualará la variable "flag_control" a 0 para que se vuelva a calcular la ecuación de la recta y se redireccione al punto de referencia. Si la variable de retorno es 0, quiere decir que el robot no se ha desviado más de 1cm, por lo que se igualará la variable "flag_control" a 2 para que el segundo controlador entre en funcionamiento.

Si "flag_control" es igual a 2, se ejecutará el control angular. Si el valor de retorno de la función "control_ang_rot" es igual a 1, significa que el robot se ha orientado en el ángulo necesario para desplazarse hacia el punto deseado, por lo que se igualará la variable "flag_control" a 3 para que entre en funcionamiento el control de posición. Caso contrario se seguirá ejecutando el control de orientación ya que el error es mayor aun a 0.01745 radianes.

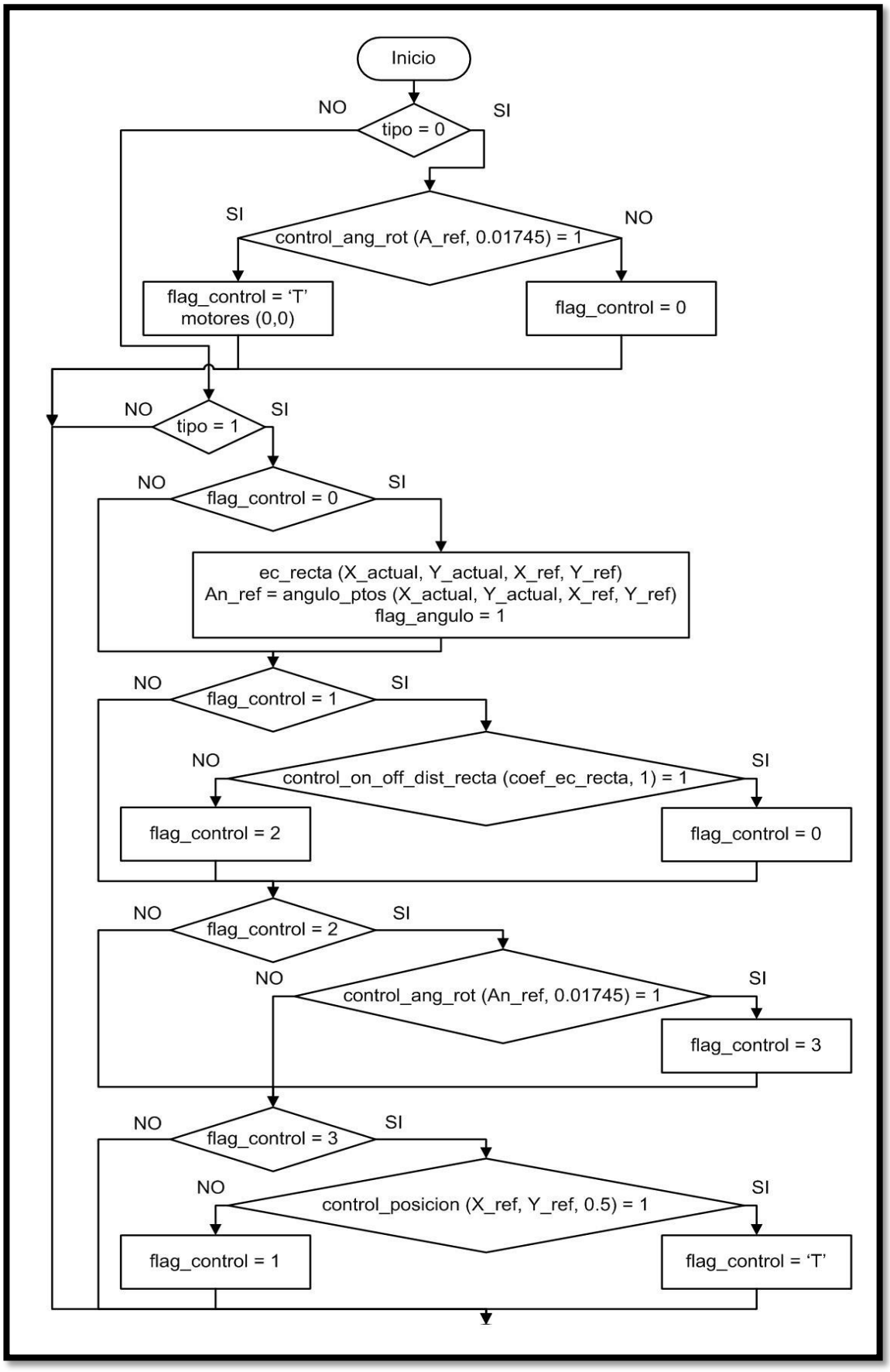
Si "flag_control" es igual a 3, se accionará el control de posición. Si el valor de retorno de la función "control_control" es igual a 1, quiere decir que el robot se encuentra a menos de 0.5 cm del punto deseado, por lo que se igualará la variable "flag_control" a 'T', indicando que se terminó la rutina de control del vector de movimiento actual de la matriz. Caso contrario se igualará "flag_control" a 1 para que verifique el control de distancia a la recta.

Una vez terminado el control en función al vector de rutina, se tiene que ejecutar el vector siguiente para generar la secuencia de vectores o coordenadas. Por lo que se preguntará si "flag_control" es igual 'T', quiere decir que se haya acabado la rutina del vector actual, y que la variable global "indice_ejecucion" (variable que determina que vector se ejecuta) sea menor que "indice_rutina" (variable que determina la cantidad de vectores). Si es menor, quiere decir que aún no se han ejecutado todos los vectores ingresados, se igualará "flag_control" y "flag_pos" a 0 para inicializar la lógica de prioridades. Además se preguntará si el siguiente vector es de tipo 2, quiere decir de espera. Si es diferente se incrementará en 1 el valor de "indice_ejecucion".

De acuerdo a la función anterior se requiere implementar un algoritmo que calcule la ecuación de la recta entre dos puntos. El diagrama de flujo de esta función se muestra en la figura 3.42. La función "ec_recta" tiene como parámetros de entrada las coordenadas de ambos puntos ("x1", "y1", "x2", "y2"). El algoritmo empieza inicializando las variables "m" (pendiente) y "b" (punto de corte del eje Y). Luego se pregunta si "x1" es igual a "x2", quiere decir que ambos puntos se encuentran en la misma recta paralela al eje Y. Si son iguales se asigna "x1" al primer elemento del vector "coef_ec_recta", los otros dos elementos serán cero. Caso contrario, se calcula "m" a través de la función "pendiente" (ver figura 3.43). Luego se halla la variable "b" como la diferencia entre "y1" y el producto de la pendiente "m" y "x1". Una vez calculados los parámetros de la recta se introducen dentro del vector "coef_ec_recta". El primer elemento será igual a la pendiente "m", el segundo elemento igual a -1 y el tercer elemento igual a "b".

En la figura 3.43 se observa el diagrama de flujo de la función "pendiente", el valor de retorno de dicha función es la división de la diferencia de "y1" y "y2" entre la diferencia entre "x1" y "x2".

control_trayectoria (tipo, X_ref, Y_ref, A_ref)



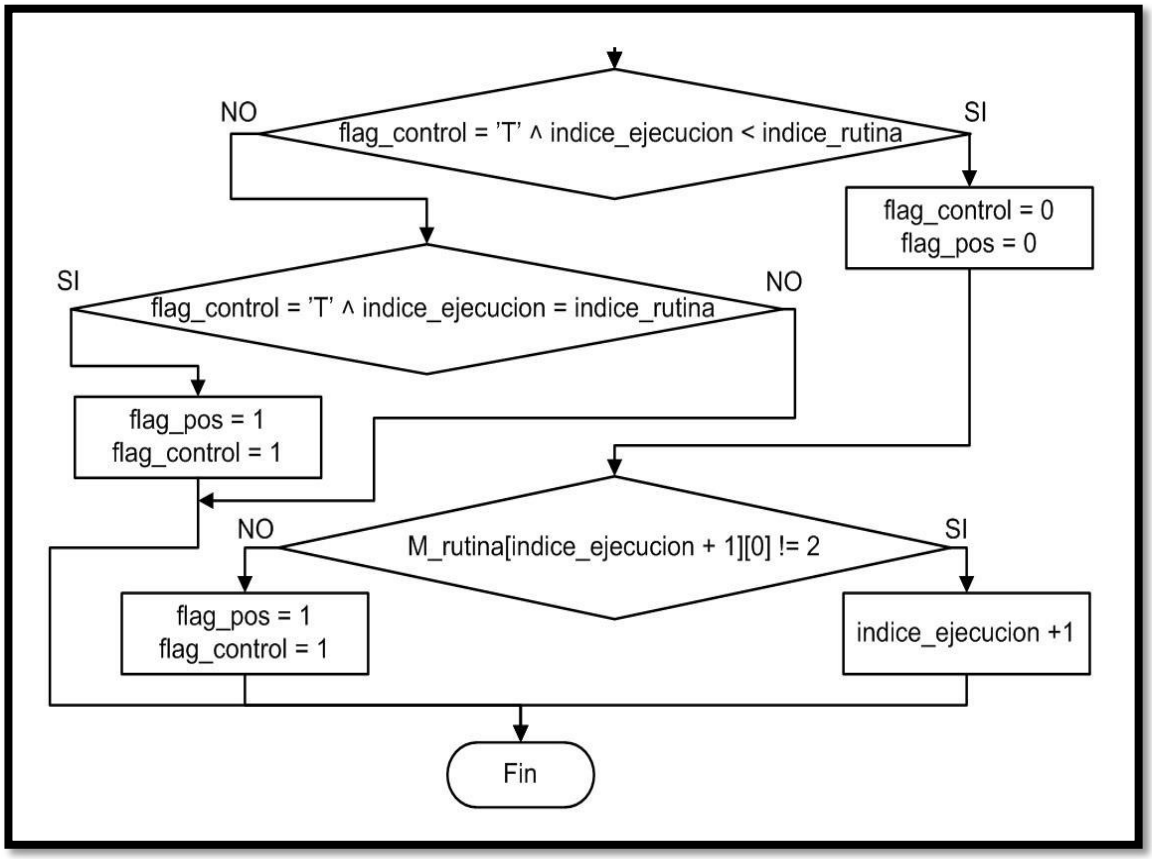


Figura 3.41.- Diagrama de flujo de la función "control_trayectoria".

ec_recta (x1, y1, x2, y2)

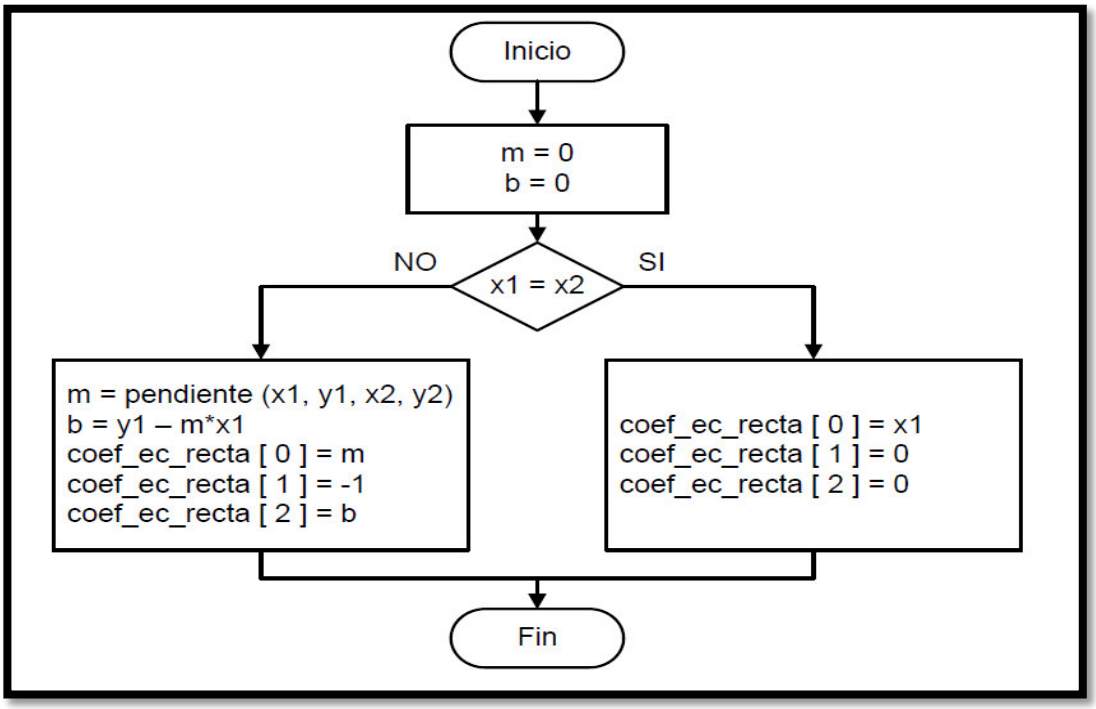


Figura 3.42.- Diagrama de flujo de la función "ec_recta".

pendiente (x1, y1, x2, y2)

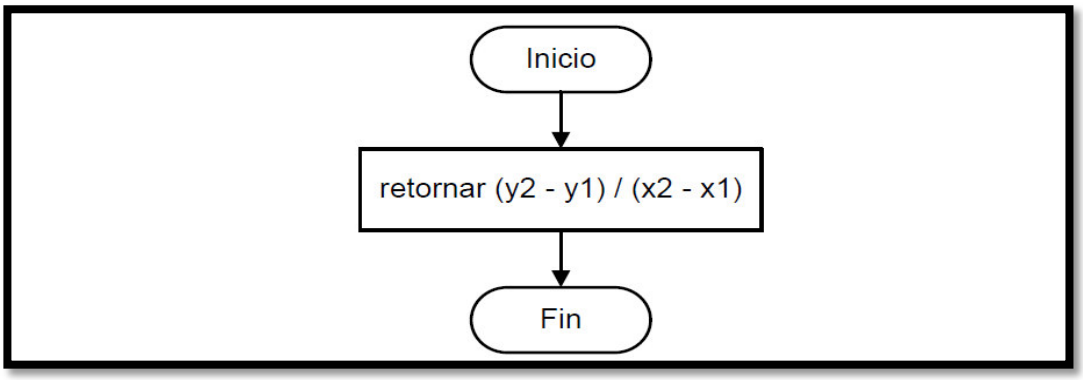


Figura 3.43.- Diagrama de flujo de la función "pendiente".

Finalmente se implementará una función que calcule el ángulo entre dos puntos respecto al eje "X". El diagrama de flujo de esta función "angulo_ptos" se observa en la figura 3.35. El algoritmo empieza inicializando la variable "ang_rlt", luego se hace una serie de preguntas para determinar si el robot se desplazará en la recta paralela del eje "X" o "Y", retornando el valor de ángulos cuadrantales. Si no se cumple ninguna de estas condiciones se calculará el valor de "ang_rlt" como el arco tangente de la pendiente de puntos iniciales y finales. Luego en función a la dirección del recorrido del robot se adicionará pi o 2pi.

angulo_ptos (x1, y1, x2, y2)

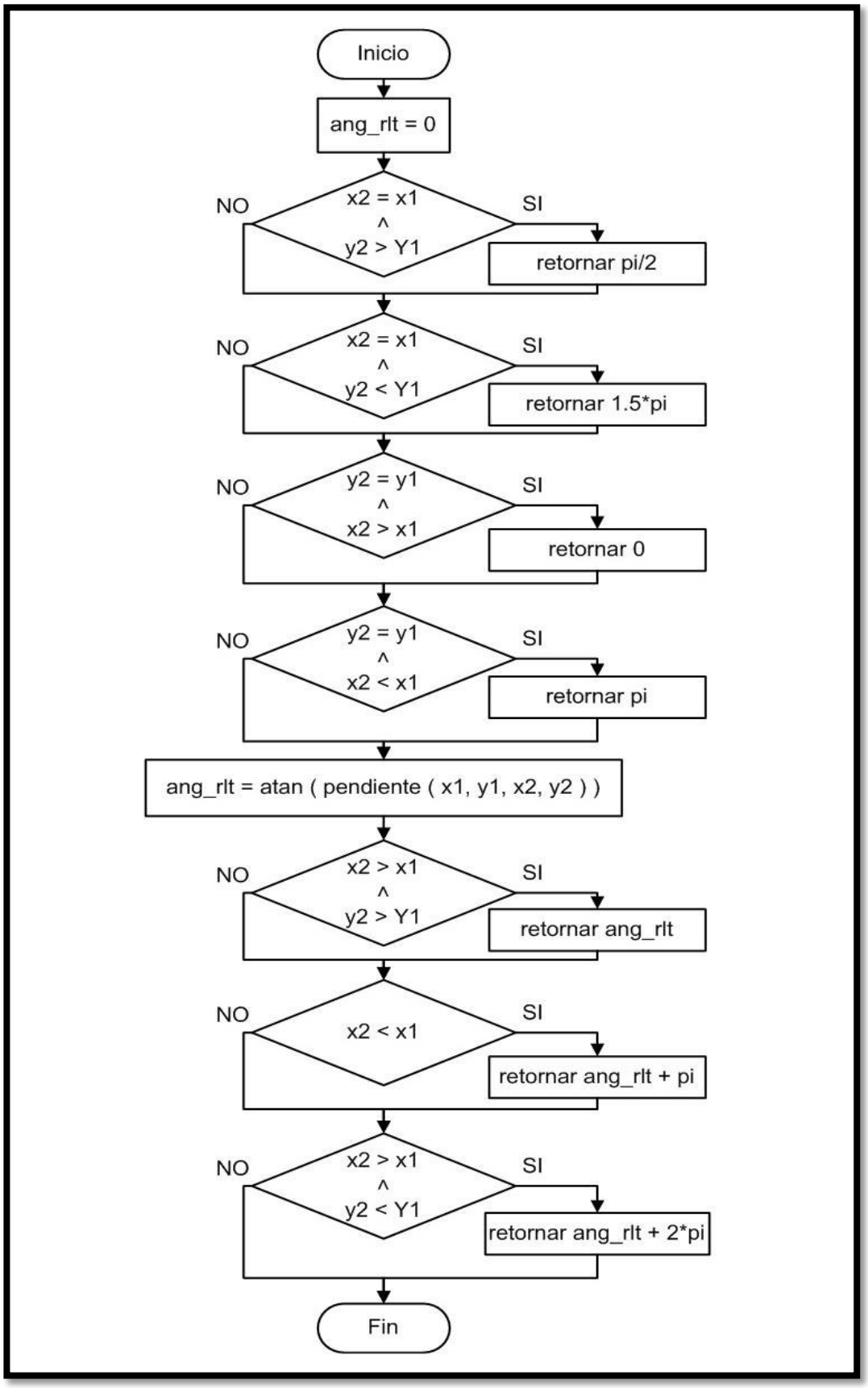


Figura 3.44.- Diagrama de flujo de la función "angulo_ptos".

La implementación de las funciones de desplazamiento en lenguaje C, se muestran a continuación.

En la línea 1 se declara e inicializa la variable global "M_rutina", la cual es una matriz de 50 por 4 que contendrá a 50 vectores de movimiento inicializados en 0. En las líneas 2 y 3 se declaran e inicializan las variables "indice_rutina" en -1 para que contabilice el vector 0 e "indice_ejecucion" en 0.

```
1.  float M_rutina[50][4]={0};
2.  int indice_rutina=-1;
3.  int indice_ejecucion=0;
```

En las líneas 4, 5, 6, 7, 8 se declaran las funciones con sus respectivos datos de entradas.

```
4.  void add_coordenadas ( float tipo,  float X_ref,  float Y_ref,  float A_ref );
5.  void control_trayectoria ( float tipo,  float X_ref,  float Y_ref,  float A_ref );
6.  void ec_recta ( float x1,  float y1,  float x2,  float y2 );
7.  float pendiente ( float x1,  float y1,  float x2,  float y2 );
8.  float angulo_ptos ( float x1,  float y1,  float x2,  float y2 );
```

En la línea 9 se desarrolla la función "add_coordenadas" del diagrama de flujo de la figura 3.39.

```
9.  void add_coordenadas ( float tipo,  float X_ref,  float Y_ref,  float A_ref )
    {
10.     indice_rutina += 1;
11.     M_rutina [ indice_rutina ][ 0 ] = tipo;
12.     M_rutina [ indice_rutina ][ 1 ] = X_ref;
13.     M_rutina [ indice_rutina ][ 2 ] = Y_ref;
14.     M_rutina [ indice_rutina ][ 3 ] = A_ref;
    }
```

En la línea 15 se desarrolla la función "control_trayectoria" del diagrama de flujo de la figura 3.41.

```
15.   void control_trayectoria(float tipo, float X_ref, float Y_ref, float A_ref)
      {
16.       if( tipo == 0 )
          {
17.           if( control_ang_rot ( A_ref, 0.01745 ) == 1 )
              {
18.               flag_control = 'T';
19.               motores(0,0);
              }
20.           else
21.               flag_control = 0;
          }
22.       else if( tipo == 1 )
          {
23.           if( flag_control == 0 )
              {
24.               ec_recta( X_actual, Y_actual, X_ref, Y_ref );
25.               An_ref = angulo_ptos( X_actual, Y_actual, X_ref, Y_ref );
26.               flag_control = 1;
              }

27.           if( flag_control == 1 )
              {
28.               If ( control_on_off_dist_recta (coef_ec_recta,1) == 1 )
29.                   flag_control = 0;
30.               else
31.                   flag_control = 2;
              }

32.           if ( flag_control == 2 )
              {
33.               if( control_ang_rot ( An_ref , 0.02 ) == 1 )
34.               flag_control = 3;
              }
```

```
35.             if( flag_control==3 )
                {
36.                 if( control_posicion(X_ref, Y_ref, 0.5)==1 )
37.                     flag_control='T';
38.                 else
39.                     flag_control=1;
                }
        }

40.     if ( flag_control == 'T' && indice_ejecucion < indice_rutina )
        {
41.         flag_control = 0;
42.         flag_pos = 0;
43.         if ( M_rutina [ indice_ejecucion + 1 ][ 0 ] != 2  )
44.             indice_ejecucion += 1;
45.         else
            {
46.             flag_pos = 1;
47.             flag_control = 1;
            }
        }
48.     else if ( flag_control == 'T'  &&  indice_ejecucion == indice_rutina )
        {
49.         flag_pos = 1;
50.         flag_control = 1;
        }
        }
```

En la línea 51 se desarrolla la función "ec_recta" del diagrama de flujo de la figura 3.42.

```
51.     void ec_recta ( float x1, float y1, float x2, float y2 )
        {
52.         float m, b;
53.         if( x1 == x2 )
            {
54.             coef_ec_recta [ 0 ] = x1;
```

```
55.             coef_ec_recta [ 1 ] = 0;
56.             coef_ec_recta [ 2 ] = 0;
            }
57.         else
            {
58.             m = pendiente ( x1, y1, x2, y2 );
59.             b = y1 - m*x1;
60.             coef_ec_recta[ 0 ] = m;
61.             coef_ec_recta[ 1 ] = -1;
62.             coef_ec_recta[ 2 ] = b;
            }
        }
```

En la línea 63 se desarrolla la función "pendiente" del diagrama de flujo de la figura 3.43.

```
63.     float pendiente(float x1, float y1, float x2, float y2)
        {
64.         return (y2-y1)/(x2-x1);
        }
```

En la línea 65 se desarrolla la función "angulo_ptos" del diagrama de flujo de la figura 3.44.

```
65.     float angulo_ptos(float x1, float y1, float x2, float y2)
        {
66.         float ang_rlt;
67.         if( x2==x1  &&  y2>y1 )   return pi/2.0;
68.         if( x2==x1  &&  y2<y1 )   return 1.5*pi;
69.         if( y2==y1  &&  x2>x1 )   return 0.0;
70.         if( y2==y1  &&  x2<x1 )   return pi;

71.         ang_rlt = atan( pendiente(x1, y1, x2, y2) );
72.         if ( x2 > x1  &&  y2 > y1  )    return ang_rlt;
73.         if ( x2 < x1                )    return ang_rlt + pi;
74.         if ( x2 > x1  &&  y2 < y1 )     return ang_rlt + 2*pi;
        }
```

Capítulo IV
Sistema de Comunicaciones

Los robots móviles deben ser monitoreados en su etapa de implementación y operación [21], si no se visualizan los parámetros del proceso en la fase de pruebas o durante la ejecución no se podrá hacer un seguimiento en tiempo real del estado del sistema. El monitoreo se puede realizar con un led, display, lcd, PC o cualquier otro dispositivo de visualización. El monitoreo mediante PC es una de las más ventajosas ya que permite visualizar los valores de los parámetros en la pantalla y dichos valores pueden ser representados mediante gráficos, cuadros, etc. Además no sólo sirve para monitorear el proceso sino también puede controlarlo mediante un programa desarrollado en el mismo. Otra de las ventajas de utilizar una PC es que podemos almacenar la información monitoreada en una base de datos o en archivo de texto.

Para monitorear un sistema externo mediante PC debe establecerse una comunicación entre ambos. En nuestro caso el robot cuenta con un microcontrolador dsPIC30F4011 con capacidad de comunicación serial, que es uno de los tipos de comunicación más usados para este propósito y además ha sido estandarizado tanto a nivel comercial como industrial en los últimos años [22].

Se requiere un programa tanto en la PC como en el microcontrolador para la transmisión y recepción de datos. En el caso de la PC el programa puede ser desarrollado en cualquier lenguaje de programación que soporte dichas características. Siendo el lenguaje Java uno de los más estables y robustos para dicha aplicación. Además del lenguaje se necesita un buen entorno de desarrollo, siendo uno de ellos el editor Eclipse, el cual posee una gran variedad de funciones para la implementación de cualquier tipo de aplicación.

Este sistema se encarga de establecer una comunicación serial bidireccional entre el robot móvil y la PC, tal como se muestra en la figura 4.1. Esta comunicación tiene como objetivo la interacción de datos en tiempo real entre ambos, de tal manera que la PC pueda mediante una interfaz gráfica registrar los parámetros del robot (ángulo, coordenadas, velocidad, etc.) y además el robot pueda recibir órdenes o parámetros (comandos de función, coordenadas y velocidades) desde la PC.

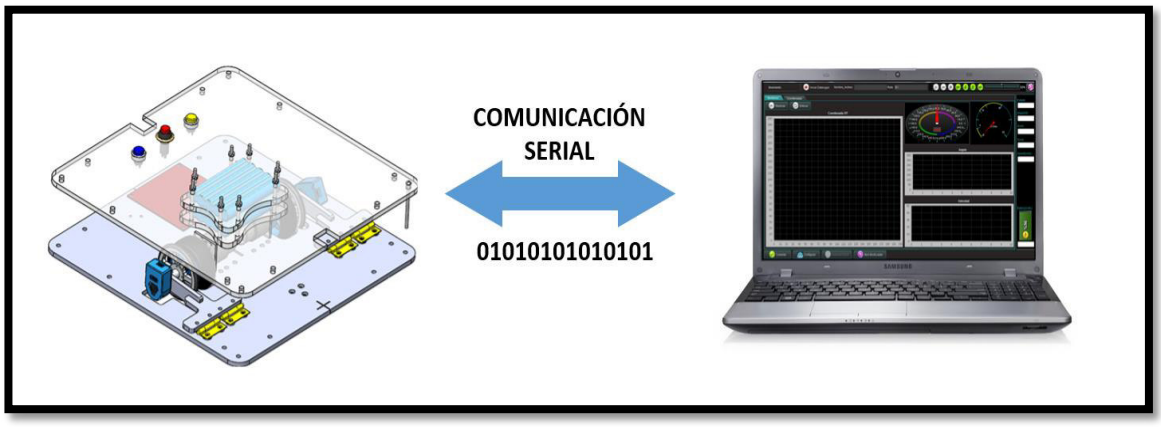


Figura 4.1.- Comunicación bidireccional entre robot y PC.

1. Comunicación del robot

Se cuenta con un robot móvil capaz de posicionarse en cualquier coordenada de un plano determinado. El robot en todo instante actualiza los valores de los parámetros (ángulo, posición, velocidad, etc.) mediante un programa de control que ejecuta el microcontrolador. Estos datos son los que se requieren para monitorear el sistema y deben ser enviados vía comunicación serial a la PC.

Características de la comunicación serial del dsPIC30F4011

El dsPIC30F4011 posee el módulo UART [23] (Universal Asynchronous Receiver Transmiter), el cual se encarga de la comunicación serial del microcontrolador. Este módulo tiene las siguientes características.

- Sistema full-dúplex asíncrono.
- Compatibilidad con interfaces RS-232.
- Datos transmisión de 8 o 9 bits.
- Paridad par, impar o ninguna (para datos de 8 bits).
- 1 ó 2 bits de parada.
- Generador de tasa de baudios con prescaler de 16 bit totalmente integrado.
- Rango de tasa de baudios entre 29bps a 1.875Mbps [24].

En la figura 4.2 se puede observar el diagrama de bloques simplificado del UART, donde el transmisor y el receptor del UART son independientes entre sí, ya que es una comunicación full-dúplex. Además ambos dependen del generador de tasa de baudios.

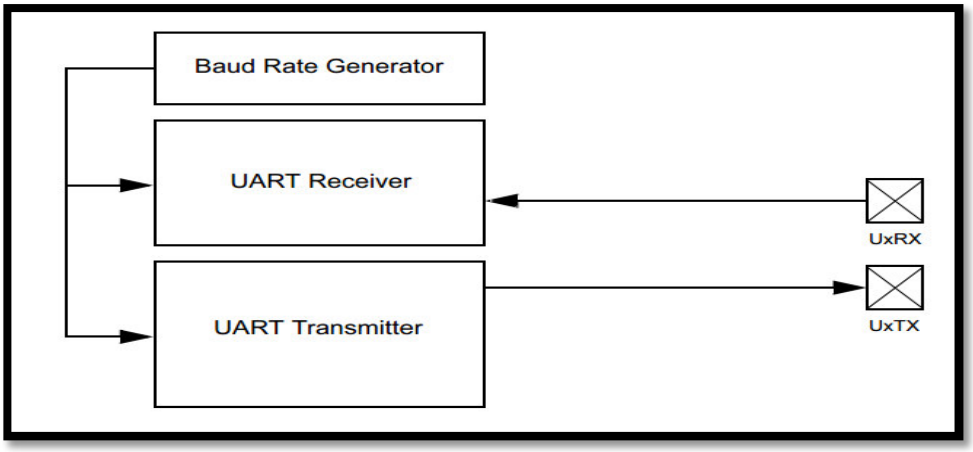


Figura 4.2.- Diagrama de bloques simplificado del UART.

Sistema de transmisión de datos

Como ya se mencionó anteriormente el robot enviará sus parámetros a la PC a través de comunicación serial. Para evitar que la PC envíe un comando predeterminado al robot asociado a un solo parámetro, se desarrolló un algoritmo para enmascarar todos los parámetros. De esta manera el robot envía todos sus datos de manera cíclica cada cierto periodo de tiempo [25] y no uno por uno, evitando de esta manera el tráfico de datos y posibles errores de sincronización. El algoritmo consiste como ya se mencionó en agrupar todos los parámetros del robot. Como todos los datos que se enviarán (ángulo, coordenada X, coordenada Y, velocidad y nivel de batería) son de tipo real, existe el inconveniente de la coma decimal debido a que estos valores serán enviados como caracteres, por lo que cada variable será multiplicada por 100 y cuando sean recibidos por la PC se dividirán entre 100 preservando dos números decimales. La estructura de la trama esquematiza con el siguiente ejemplo.

SMA10000X05000Y07000V0500B1300FIEE

A10000	Angulo	= 100.00 grados
X05000	X	= 50.00 cm
Y07000	Y	= 70.00 cm
V0500	Velocidad	= 5.00 m/s
B1300	Batería	= 13.00 volts

Donde **SM A X Y V B FIEE** son los caracteres de validación de trama, los cuales serán verificados al momento de recibir la trama por la PC. Se creó la función "empaquetador_datos", la cual ejecuta está lógica de enmascaramiento de datos. El diagrama de flujo de dicho algoritmo se muestra en la figura 4.3.

El algoritmo empieza inicializando las variables "x", "y", "angulo" y "v". Además se declara el vector de 37 caracteres "trama". El primer paso consiste en calcular el valor de las variables mencionadas asignándoles los valores de sus respectivos valores globales y multiplicándolos por 100. En el caso del ángulo se realiza una pequeña conversión de radianes a grados sexagesimales. El segundo paso es preguntar si el ángulo es negativo, si lo fuera se le agregará 36000 (360 x 100) para volverlo positivo. El tercer paso consiste en agregar los elementos que conformarán la trama caracter por caracter. Como ya se mencionó la trama contendrá además de los valores de los parámetros del robot caracteres de validación de trama, los cuales indicarán cuando empieza una trama y cuando termina, de igual manera servirá para determinar si una trama ha sido enviada correctamente verificando que cada caracter de validación en el lugar que le corresponde. La trama inicia con dos caracteres de validación "S" y "M" en la posición 0 y 1 respectivamente. En la posición 2 se coloca otro caracter de validación "A" indicando que los elementos después de este pertenecen a la orientación angular del robot. Para que el envío de los números sea coherente, quiere decir que siempre contenga 5 caracteres se descompuso cada número dígito a dígito. Además se le sumará el caracter '0', para convertir justamente los datos numéricos a caracteres. El mismo principio se utilizará para la adición de los otros parámetros (coordenada X, coordenada Y, velocidad y nivel de batería). Finalmente se agregará los caracteres de validación finales "F", "I", "E" y "E", así como también el return o enter "\r" y la nueva línea "\n". El cuarto y último paso consiste en enviar el vector trama a través del puerto serial usando la función "stringUART".

empaquetador_datos ( )

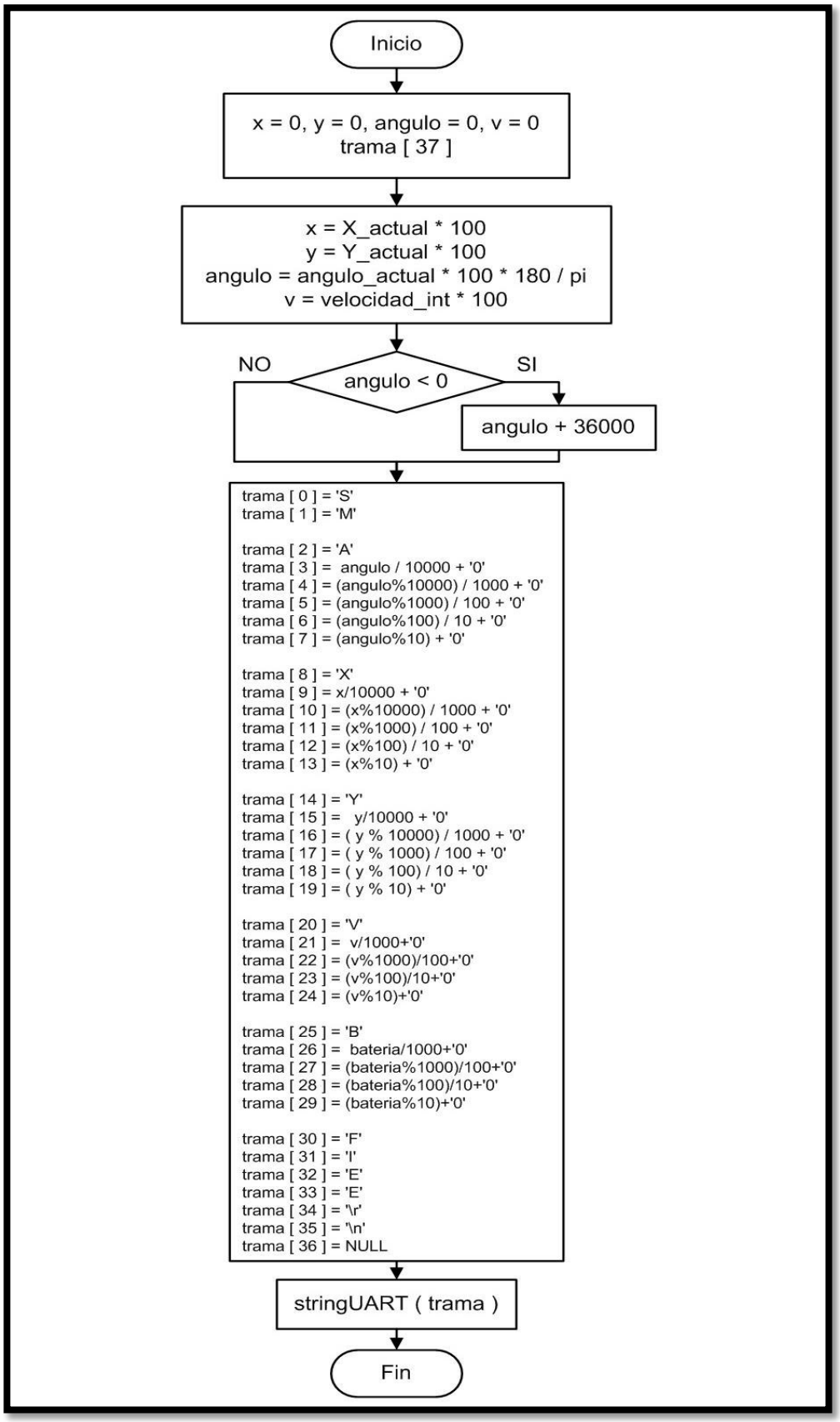


Figura 4.3.- Diagrama de bloques de la función "empaquetador_datos".

Sistema de recepción de datos

Así como el robot tiene la capacidad de enviar sus parámetros a través de una trama de información, también puede recibir datos provenientes de la PC. Como ya se mencionó el robot emplea los vectores de la matriz "M_rutina" para poder posicionarse en distintos puntos de manera secuencial. Por lo que la PC enviará dichos vectores para que sean agregados a la matriz y de esta manera poder agregar puntos a los cuales se dirigirá el robot en línea o durante su operación. La función que ejecutará esta lógica se llama "recibir_cord". El algoritmo empieza inicializando las variables "tipo", "X", "Y" y "A" en 0. Luego se declara "val_car", la cual se igualará a la diferencia entre la variable global "RcvCar" (esta variable posee el caracter recibido a través del puerto serial del dsPIC) y el caracter '0' (48 en decimal). Se le resta este carácter para convertir los datos ASCII a decimales.

La estructura de la trama de recepción de coordenadas se esquematiza con el siguiente ejemplo.

C1100200270**P**

1	tipo	=	tipo traslación
100	X	=	100 cm
200	Y	=	200 cm
270	Ángulo	=	270 grados

Donde C y P son los caracteres de inicio y fin de la cadena respectivamente.

La trama está conformada por cuatro parámetros que se agregarán a la matriz "M_rutina". El primer elemento es el tipo de movimiento (0 para rotación, 1 para traslación y 2 para espera). El segundo, tercer y cuarto elemento están conformados por 3 dígitos (centena, decena y unidad) que forman la coordenada "X", la coordenada "Y" y el ángulo de referencia.

Continuando con el algoritmo, se preguntará si el valor de "val_car" es igual a 19; ya que el caracter 'C' es igual a 67 en decimal y si le restamos '0' (48 en decimal) obtendremos 19; y si "flag_cord" es igual a 0. Si se cumple la condición, se iguala "flag_cord" a 1, para que en el siguiente ciclo se ejecute otra condicional. Caso contrario, se preguntará si "flag_cord" es igual 1, "val_car" positivo (mayor a cero) y "val_car" menor igual a nueve (solo se aceptarán dígitos de 0 a 9). Si se cumple esta condición se igualará el elemento del vector "data", tomando como índice la variable "indice_llegada_cord", al dígito numérico "val_car" y se aumentará en 1 en valor del

índice "indice_llegada_cord". Caso contrario, se preguntará si "flag_cord" es igual 1, si "val_car" es igual a 32 (ya que el caracter ‘P’ es igual a 80 en decimal y si le restamos ‘0’ (48 en decimal) obtendremos 32) y si "indice_llegada_cord" es igual a 10 (se recibirán los nueve caracteres numéricos correspondientes a la coordenada "X", coordenada "Y" y el ángulo de orientación). Si se cumple esta condición, se procederá a decodificar la trama caracter por caracter en función a la estructura de la trama. Se extraerá el tipo de movimiento, la coordenada X, la coordenada Y y el ángulo, una vez decodificado se ejecutará la función "add_coordenadas" ingresando los datos obtenidos. Finalmente se iguala "indice_llegada_cord" y "flag_cord" a 0 para que el sistema esté listo para una nueva recepción. De esta manera se puede enviar coordenadas a través de la PC cuando el robot está en operación.

recibir_cord ( )

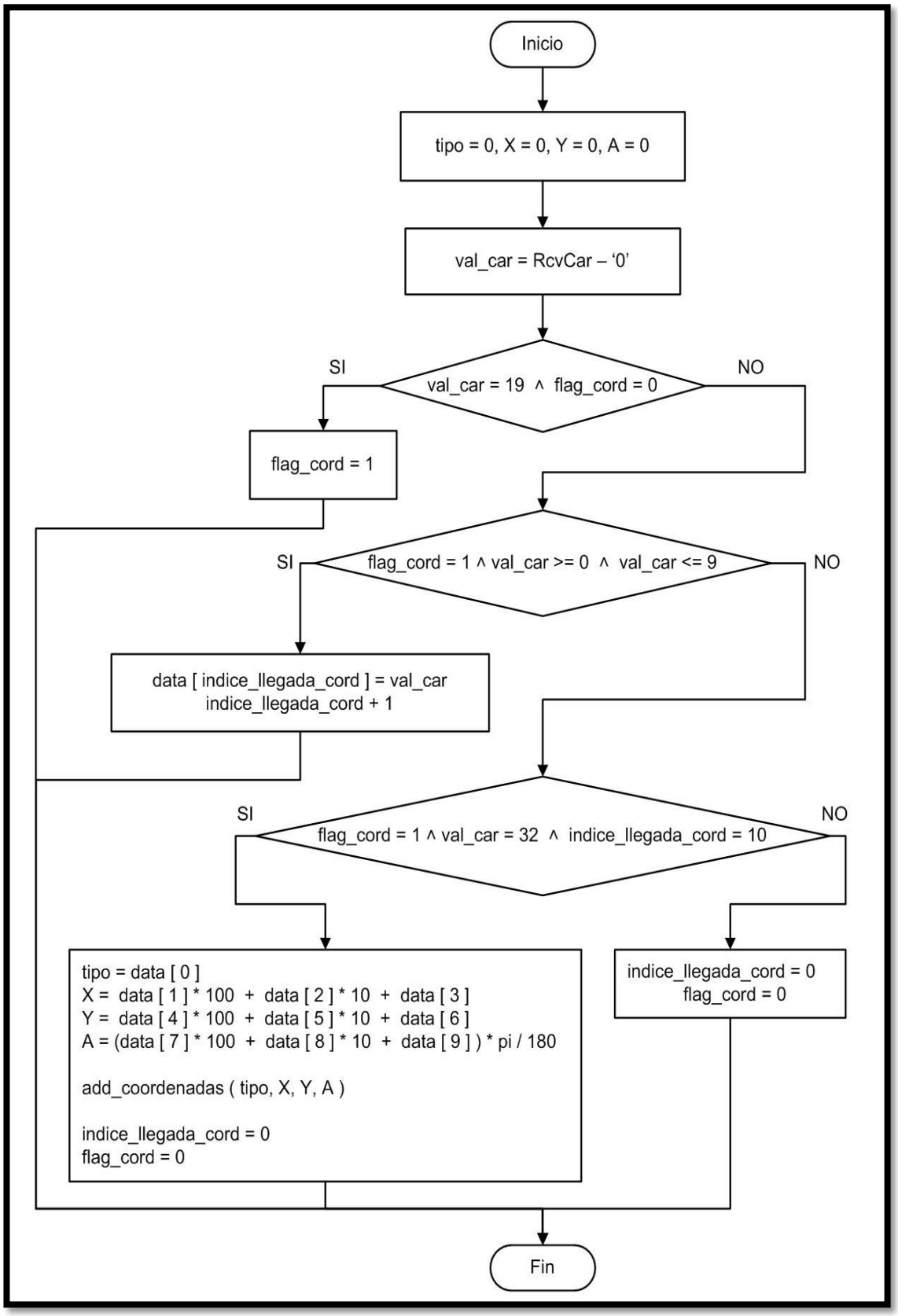


Figura 4.4.- Diagrama de bloques de la función "recibir_cord".

La implementación de las funciones de comunicación de transmisión de trama y recepción de coordenadas en lenguaje C se muestra a continuación.

En la línea 1 se declara la variable "RcvCar" la cual contendrá el valor del caracter recibido a través del puerto serial del dsPIC. En la línea 2 se declara el vector "data" el cual alojará a todos los caracteres que irán llegando durante la recepción de las coordenadas. Así también en las líneas 3 y 4 se declararán e inicializarán las variables "flag_cord" e "indice_llegada_cord" en 0.

```
1.  char RcvCar;
2.  char data [ 20 ];
3.  int flag_cord = 0;
4.  int indice_llegada_cord = 0;
```

En las líneas 5 y 6 se declaran las funciones "empaquetador_datos" y "recibir_crod".

```
5.  void empaquetador_datos (void);
6.  void recibir_cord (void);
```

A partir de la línea 7 hasta la 53 se desarrolla la función "empaquetador_datos" del algoritmo de la figura 4.3.

```
7.  void empaquetador_datos()
    {
8.      long x, y, angulo,v;
9.      char trama[37];
10.     x = X_actual*100;
11.     y = Y_actual*100;
12.     angulo = angulo_actual * 100 * 180 / pi;
13.     v = velocidad_inst * 100;

14.     if ( angulo < 0 )
15.         angulo = angulo + 36000;

16.     trama[0] = 'S';
17.     trama[1] = 'M';
```

```
18.             trama[ 2 ] = 'A';
19.             trama[ 3 ] =   angulo / 10000 + '0';
20.             trama[ 4 ] = ( angulo % 10000 ) / 1000 + '0';
21.             trama[ 5 ] = ( angulo % 1000 ) / 100 + '0';
22.             trama[ 6 ] = ( angulo % 100 ) / 10 + '0';
23.             trama[ 7 ] = ( angulo % 10 ) + '0';

24.             trama[ 8 ] = 'X';
25.             trama[ 9 ] = x / 10000 + '0';
26.             trama[ 10 ] = ( x % 10000 ) / 1000 + '0';
27.             trama[ 11 ] = ( x % 1000 ) / 100 + '0';
28.             trama[ 12 ] = ( x % 100 ) / 10 + '0';
29.             trama[ 13 ] = ( x % 10 ) + '0';

30.             trama[ 14 ] = 'Y';
31.             trama[ 15 ] = y / 10000 + '0';
32.             trama[ 16 ] = ( y % 10000 ) / 1000 + '0';
33.             trama[ 17 ] = ( y % 1000 ) / 100 + '0';
34.             trama[ 18 ] = ( y%100 ) / 10 + '0';
35.             trama[ 19 ] = ( y % 10 ) + '0';

36.             trama[ 20 ] = 'V';
37.             trama[ 21 ] = v / 1000 + '0';
38.             trama[ 22 ] = ( v % 1000 ) / 100 + '0';
39.             trama[ 23 ] = ( v % 100 ) / 10 + '0';
40.             trama[ 24 ] = ( v % 10 ) + '0';

41.             trama[ 25 ] = 'B';
42.             trama[ 26 ] =   bateria / 1000 + '0';
43.             trama[ 27 ] = ( bateria % 1000 ) / 100 + '0';
44.             trama[ 28 ] = ( bateria % 100 ) / 10 + '0';
45.             trama[ 29 ] = ( bateria % 10 ) + '0';

46.             trama[ 30 ] = 'F';
47.             trama[ 31 ] = 'I';
48.             trama[ 32 ] = 'E';
49.             trama[ 33 ] = 'E';
50.             trama[ 34 ] = '\r';
51.             trama[ 35 ] = '\n';
52.             trama[ 36 ] = NULL;
```

```
53.             stringUART ( trama );
        }
```

A partir de la línea 54 hasta la 72 se desarrolla la función "recibir_cord" del algoritmo de la figura 4.4.

```
54.     void recibir_cord(void)
        {
55.       int val_car = RcvCar - '0';
56.       int tipo;

57.       If ( val_car == 19 && flag_cord == 0 )
58.             flag_cord = 1;
59.       else if ( flag_cord == 1  &&  val_car >= 0  &&  val_car <= 9 )
          {
60.             data [ indice_llegada_cord ] = val_car;
61.             indice_llegada_cord++;
          }
62.       else if ( flag_cord == 1  &&  val_car == 32   &&  indice_llegada_cord == 10 )
          {
63.         tipo = data[ 0 ];
64.         X = data[ 1 ] * 100.0  +  data[ 2 ] * 10.0  +  data[3];
65.         Y = data[ 4 ] * 100.0  +  data[ 5 ] * 10.0  +  data[6];
66.         A = ( data[ 7 ] * 100.0  +  data[ 8 ] * 10.0  +  data[ 9 ]  ) * pi / 180.0;

67.         add_coordenadas( tipo , X , Y , A);
68.         indice_llegada_cord = 0;
69.         flag_cord=0;
          }
70.       else
          {
71.         indice_llegada_cord = 0;
72.         flag_cord=0;
          }
        }
```

2. Interfaz de monitoreo

Para realizar el monitoreo del sistema durante el funcionamiento del robot se utilizó la PC, en la cual se desarrolló una aplicación gráfica en Java [26]. La cual permite visualizar las variables del sistema (coordenadas, orientación, velocidad, secuencia de puntos, etc) a través de gráficas y casillas de texto. Se decidió utilizar Java como plataforma de desarrollo debido a que se ejecuta sobre una máquina virtual, quiere decir que no depende del hardware de la PC, sino del software. Esta característica es sumamente importante debido a que la aplicación podrá ejecutarse en cualquier PC sin importar si es de 32 o 64 bits o inclusive sin importar su sistema operativo. Otra de las ventajas de trabajar con Java es que posee una gran cantidad de librerías con las que se puede desarrollar prácticamente cualquier aplicación. Además Java permite la programación orientada a objetos (POO) lo cual facilitó enormemente la codificación ya que se crearon clases específicas para cada tipo de tarea (adquisición, ajuste de data, datalogger, GUI, etc.).

Para el desarrollo de la aplicación se emplearon tres librerías. La primera es JavaSE-1.7, es la librería por defecto de java, con la cual se implementó la interfaz gráfica (GUI) y la herramienta de datalogging (historizar o grabar ciertas variables en una hoja de texto). La segunda librería es la RXTX, esta librería permite la interacción entre la PC y su puerto serial, con la cual es posible la comunicación con el robot. Por último la tercera librería es la JFreeChart, la cual contiene distintos tipos de gráficos (Gráfica XY, dial, meter, etc.) permitiendo poder mostrar los parámetros del robot de manera gráfica.

Las tareas que se desarrollarán empleando las 3 librerías generan un tiempo de procesamiento del computador, por los que a más tareas el tiempo de proceso aumentará generando que la aplicación sea más lenta. Para evitar este problema se emplearon threads. Los threads o hilos permiten al computador desarrollar distintas tareas al mismo tiempo. Los distintos hilos de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, situación de autenticación, etc. Esta técnica permite simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente. Un hilo es simplemente una tarea que puede ser ejecutada al mismo tiempo que otra tarea. Los hilos de ejecución que comparten los mismos recursos, sumados a estos recursos, son en conjunto

conocidos como un proceso. El hecho de que los hilos de ejecución de un mismo proceso compartan los recursos hace que cualquiera de estos hilos pueda modificar éstos. Cuando un hilo modifica un dato en la memoria, los otros hilos acceden a ese dato modificado inmediatamente.

En la figura 4.5 se observa el diagrama de la interfaz de monitoreo, cada tarea está asociada a una librería específica, y estas a su vez a un thread, de tal manera que cada tarea se ejecute en paralelo reduciendo el procesamiento de un solo procesador y generando una aplicación en real time.

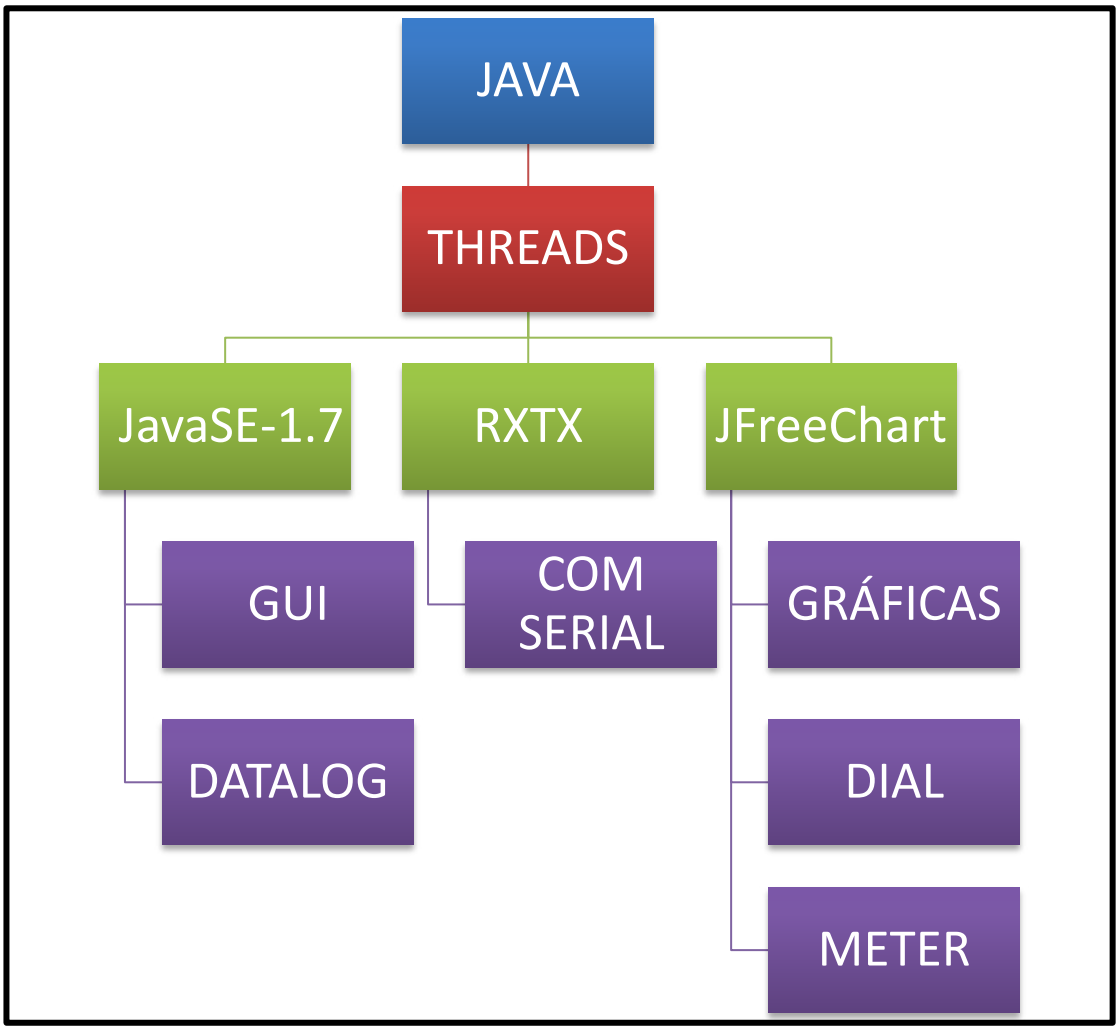


Figura 4.5.- Diagrama de la interfaz de monitoreo.

La interfaz cuenta con las siguientes herramientas y aplicativos.

Ventana de configuración del puerto serial

Esta ventana permite configurar los distintos parámetros del puerto serial de la PC, como el Baud Rate (velocidad de transmisión), Data bits, Stop bits y la paridad. En la figura 4.6 se muestran los parámetros con los cuales trabaja la interfaz para comunicarse con el robot. Además esta ventana cuenta con una opción de identificación automática de puertos disponibles en la PC. Esta herramienta es muy útil debido a que se emplean conversores de puerto serial a USB, los cuales asignan un puerto COM de manera aleatoria, el cual muchas veces es difícil de identificar.

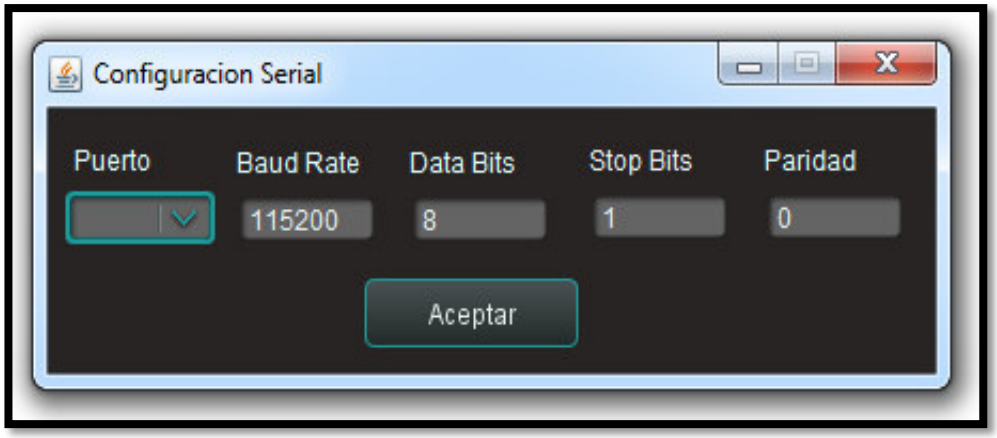


Figura 4.6.- Ventana de configuración del puerto serial.

Ventana principal

Esta ventana (ver figura 4.7) contiene todos los elementos de monitoreo y supervisión tanto de forma gráfica como numérica. Además poseen los botones para distintas aplicaciones como el datalog, bootloader, configuración serial, etc. Así mismo, cuenta con una sub-pantalla en la cual se configura y codifica la trama para el envío de coordenadas al robot.

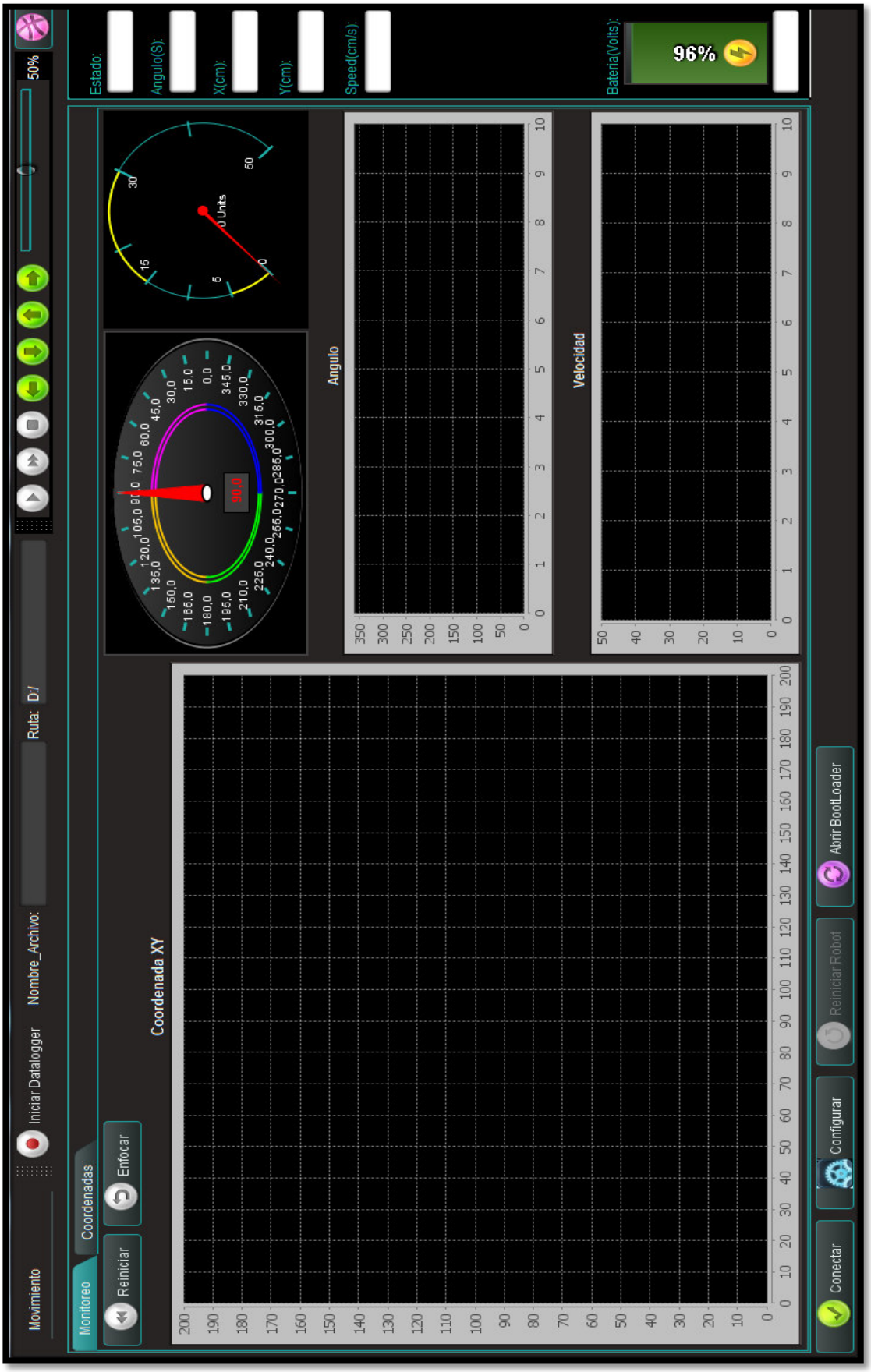


Figura 4.7.- Ventana principal.

Gráfica de coordenadas XY

La interfaz cuenta con un área (ver figura 4.8) en la cual se plotea en tiempo real la posición del robot respecto a sus coordenadas "XY".

Figura 4.8.- Grafica de coordenadas XY.

Gráfica de ángulo

Se cuenta con un área (ver figura 4.9) en la cual se grafica en tiempo real la orientación del robot respecto al eje X.

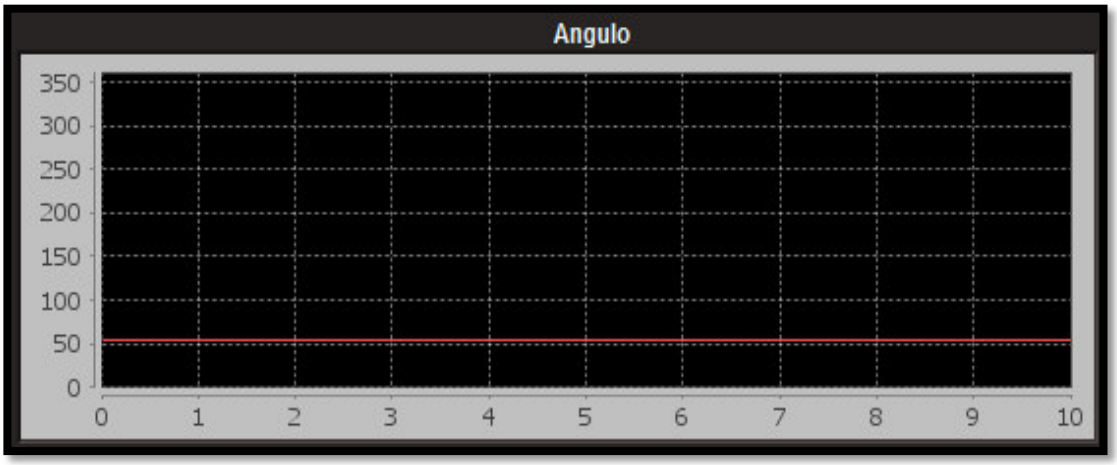


Figura 4.9.- Grafica de ángulo.

Gráfica de velocidad

La velocidad también es graficada en tiempo real, tal como se muestra en la figura 4.10.

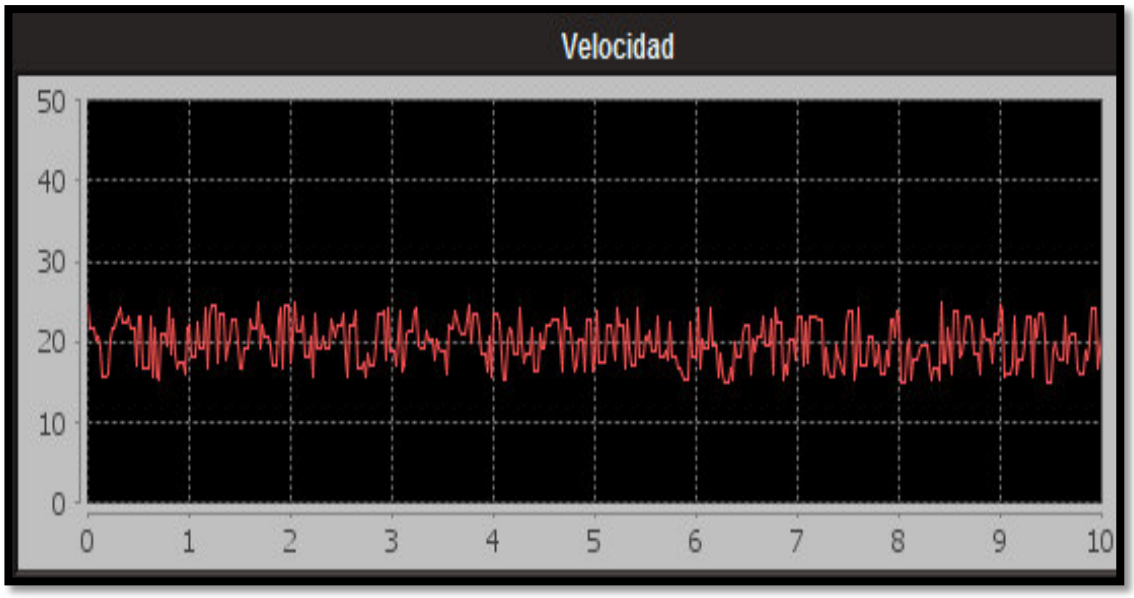


Figura 4.10.- Grafica de velocidad.

Dial y Meter de orientación y velocidad

Para una mejor visualización de la orientación angular y la velocidad se emplearon el dial y el meter como elementos gráficos de monitoreo (ver figura 4.11). Con el Dial se tiene una idea más acertada de la orientación del robot, debido a que funciona como una especie de brújula indicando en todo momento hacia donde se dirige el robot.

Figura 4.11.- Dial y Meter de orientación y velocidad.

Cuadro de parámetros

En este cuadro (figura 4.12) de muestran los parámetros del robot (estado, ángulo, coordenada "X", coordenada "Y" y la velocidad).

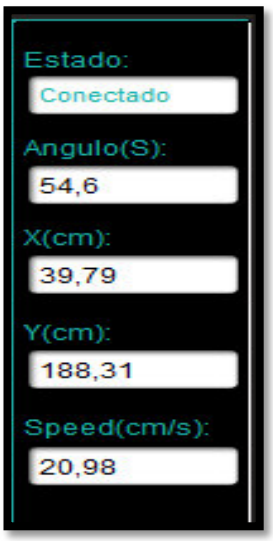


Figura 4.12.- Cuadro de parámetros.

Nivel de batería

Uno de los parámetros más importantes a tener en cuenta en un robot autónomo es su nivel de energía. Por lo que se requiere un monitoreo constante del nivel de tensión de la batería que alimenta el sistema. La interfaz cuenta con una barra que muestra de manera proporcional el nivel y porcentaje de voltaje.

Figura 4.13.- Nivel de batería.

Datalogger

El datalogger es un registro histórico que se realiza a una o cierto grupo de variables. Esta herramienta es muy útil al momento de hacer un estudio de los distintos parámetros durante una operación. La barra en la cual se configura el nombre del archivo que se creará y la ruta del mismo se observa en la figura 4.14. El archivo generado es de tipo .txt (ver figura 4.15).

Figura 4.14.- Ventana principal.

Archivo Edición Formato Ver Ayuda

Proyecto:Robot para el Transporte de Productos con Control de Trayectora Mediante Principios Odometricos
Universidad Nacional Mayor de San Marcos
Facultad de Ingenieria Electronica

Parametros Muestreados

Time	X	Y	Angulo	Velocidad	Bateria	Contador
15.13.51.655	X: 087,400	Y: 111,500	A: 194,530	V: 019,209	B: 012,572	0
15.13.51.678	X: 087,400	Y: 111,500	A: 194,530	V: 023,205	B: 012,488	0
15.13.51.693	X: 087,400	Y: 111,500	A: 194,530	V: 017,841	B: 011,813	0
15.13.51.724	X: 087,400	Y: 111,500	A: 194,530	V: 016,375	B: 012,545	0
15.13.51.741	X: 087,400	Y: 111,500	A: 194,530	V: 022,865	B: 011,301	0
15.13.51.772	X: 087,400	Y: 111,500	A: 194,530	V: 015,826	B: 011,278	0
15.13.51.788	X: 087,400	Y: 111,500	A: 194,530	V: 016,585	B: 011,313	0
15.13.51.820	X: 087,400	Y: 111,500	A: 194,530	V: 019,427	B: 011,987	0
15.13.51.836	X: 087,400	Y: 111,500	A: 194,530	V: 024,243	B: 012,067	0
15.13.51.868	X: 087,400	Y: 111,500	A: 194,530	V: 015,708	B: 011,427	0
15.13.51.884	X: 087,400	Y: 111,500	A: 194,530	V: 022,389	B: 011,179	0
15.13.51.916	X: 087,400	Y: 111,500	A: 194,530	V: 019,823	B: 011,044	0
15.13.51.932	X: 087,400	Y: 111,500	A: 194,530	V: 020,192	B: 012,847	0
15.13.51.964	X: 087,400	Y: 111,500	A: 194,530	V: 018,296	B: 011,894	0
15.13.51.980	X: 087,400	Y: 111,500	A: 194,530	V: 016,058	B: 012,343	0
15.13.51.996	X: 087,400	Y: 111,500	A: 194,530	V: 021,644	B: 012,552	0
15.13.52.028	X: 087,400	Y: 111,500	A: 194,530	V: 021,728	B: 011,713	0
15.13.52.044	X: 087,400	Y: 111,500	A: 194,530	V: 017,684	B: 011,610	0
15.13.52.076	X: 087,400	Y: 111,500	A: 194,530	V: 016,075	B: 011,390	0
15.13.52.092	X: 087,400	Y: 111,500	A: 194,530	V: 017,271	B: 012,259	0
15.13.52.124	X: 087,400	Y: 111,500	A: 194,530	V: 021,268	B: 011,884	0
15.13.52.140	X: 087,400	Y: 111,500	A: 194,530	V: 018,578	B: 011,873	0
15.13.52.172	X: 087,400	Y: 111,500	A: 194,530	V: 024,628	B: 011,634	0
15.13.52.188	X: 087,400	Y: 111,500	A: 194,530	V: 015,606	B: 012,531	0
15.13.52.220	X: 087,400	Y: 111,500	A: 194,530	V: 020,587	B: 011,672	0
15.13.52.235	X: 087,400	Y: 111,500	A: 194,530	V: 021,563	B: 012,972	0
15.13.52.267	X: 087,400	Y: 111,500	A: 194,530	V: 018,440	B: 012,831	0
15.13.52.283	X: 087,400	Y: 111,500	A: 194,530	V: 017,995	B: 012,584	0
15.13.52.315	X: 087,400	Y: 111,500	A: 194,530	V: 023,280	B: 012,898	0
15.13.52.331	X: 087,400	Y: 111,500	A: 194,530	V: 016,097	B: 011,766	0
15.13.52.363	X: 087,400	Y: 111,500	A: 194,530	V: 019,436	B: 011,318	0
15.13.52.379	X: 087,400	Y: 111,500	A: 194,530	V: 017,514	B: 012,194	0
15.13.52.395	X: 087,400	Y: 111,500	A: 194,530	V: 018,020	B: 011,851	0
15.13.52.427	X: 087,400	Y: 111,500	A: 194,530	V: 018,128	B: 011,389	0
15.13.52.443	X: 087,400	Y: 111,500	A: 194,530	V: 022,789	B: 012,927	0
15.13.52.475	X: 087,400	Y: 111,500	A: 194,530	V: 019,783	B: 012,326	0
15.13.52.491	X: 087,400	Y: 111,500	A: 194,530	V: 021,402	B: 011,350	0
15.13.52.523	X: 087,400	Y: 111,500	A: 194,530	V: 021,775	B: 012,786	0
15.13.52.539	X: 087,400	Y: 111,500	A: 194,530	V: 022,389	B: 011,471	0
15.13.52.571	X: 087,400	Y: 111,500	A: 194,530	V: 015,976	B: 012,631	0
15.13.52.587	X: 087,400	Y: 111,500	A: 194,530	V: 021,664	B: 012,991	0

Figura 4.15.- Archivo de texto generado por el datalogger.

Ventana de envío de coordenadas

Se cuenta con una pantalla para seleccionar el tipo de movimiento y los valores de setpoint de la coordenada "X", coordenada "Y" y el ángulo deseado. Cuando se presiona el botón "Agregar" se ejecuta un algoritmo que codifica los parámetros y los convierte en una trama que se enviará vía comunicación serial al robot.

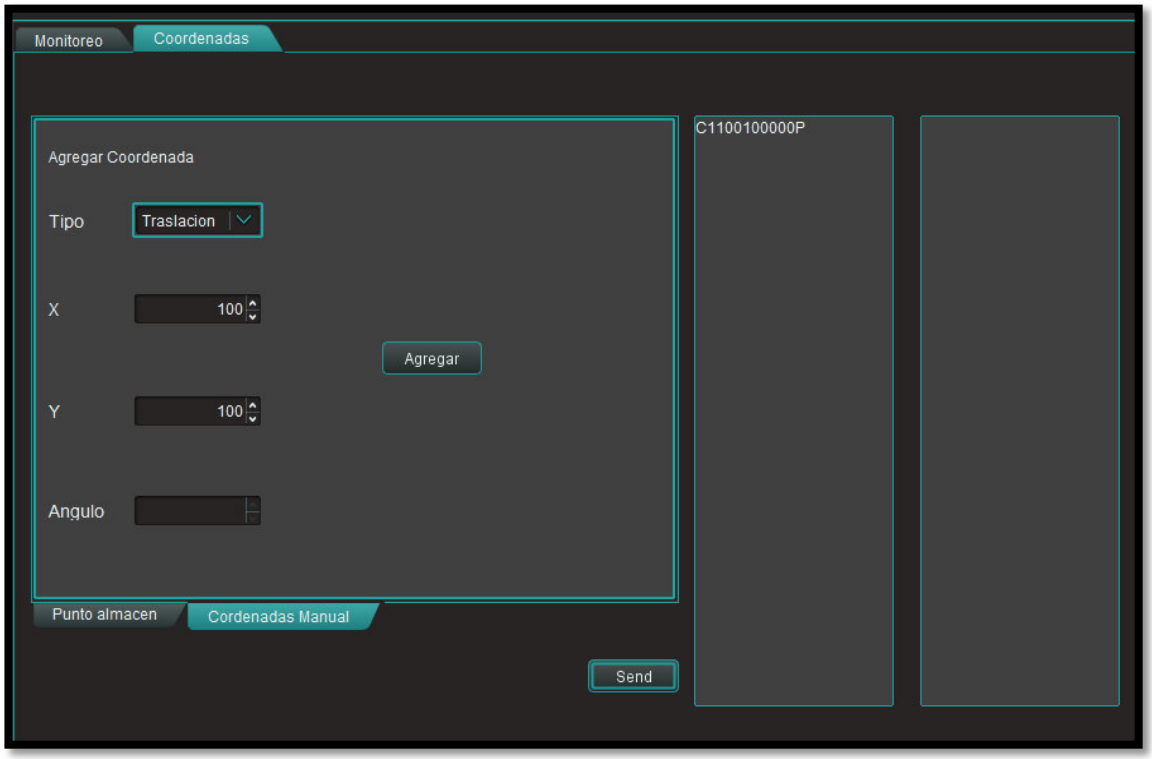


Figura 4.16.- Ventana de envío de coordenadas.

Capítulo V
Resultados

En este capítulo se presentan los resultados obtenidos en las pruebas de lectura de encoders, análisis odométrico y el sistema de control. Se realizaron 13 pruebas en total para comprobar el correcto funcionamiento sistema, tanto a nivel de controladores discretos como de algoritmos de control y trayectoria. Para la toma de datos se utilizó la interfaz gráfica desarrollada en Java, la cual capturó los datos y los ploteó en gráficas de tiempo real, permitiendo exportarlos para su posterior análisis. Las pruebas se llevaron a cabo en un ambiente controlado, el cual cuenta con una superficie plana sin desniveles y libre de obstáculos.

Prueba 1: Lectura de encoders

Esta prueba demuestra el correcto funcionamiento del algoritmo de lectura de encoders a través del conteo de ticks positivos y negativos. La cantidad de ticks obtenidos se comparará con el número de vueltas realizas en el eje del encoder, el cual posee 500 ticks por revolución (capítulo 1, 2.Encoder).

La prueba consiste en girar la rueda que está acoplada al eje del encoder, la cual se muestra en la figura 5.1. Esta rueda posee una marca para referenciar el fin de una revolución completa.

Los resultados se presentan en la tabla 5.1, donde se muestra la cantidad de ticks contados en sentido horario como anti horario. Los cuales presentan un error mínimo en comparación a los ticks esperados. Incluso luego de realizar 100 revoluciones el error detectado fue de 5 ticks de 50000, esto equivale a 3.6º en el eje del encoder. Esta medida convertida a un desplazamiento lineal de la rueda equivale a 0.21 cm, el cual es un error aceptable para nuestra aplicación.

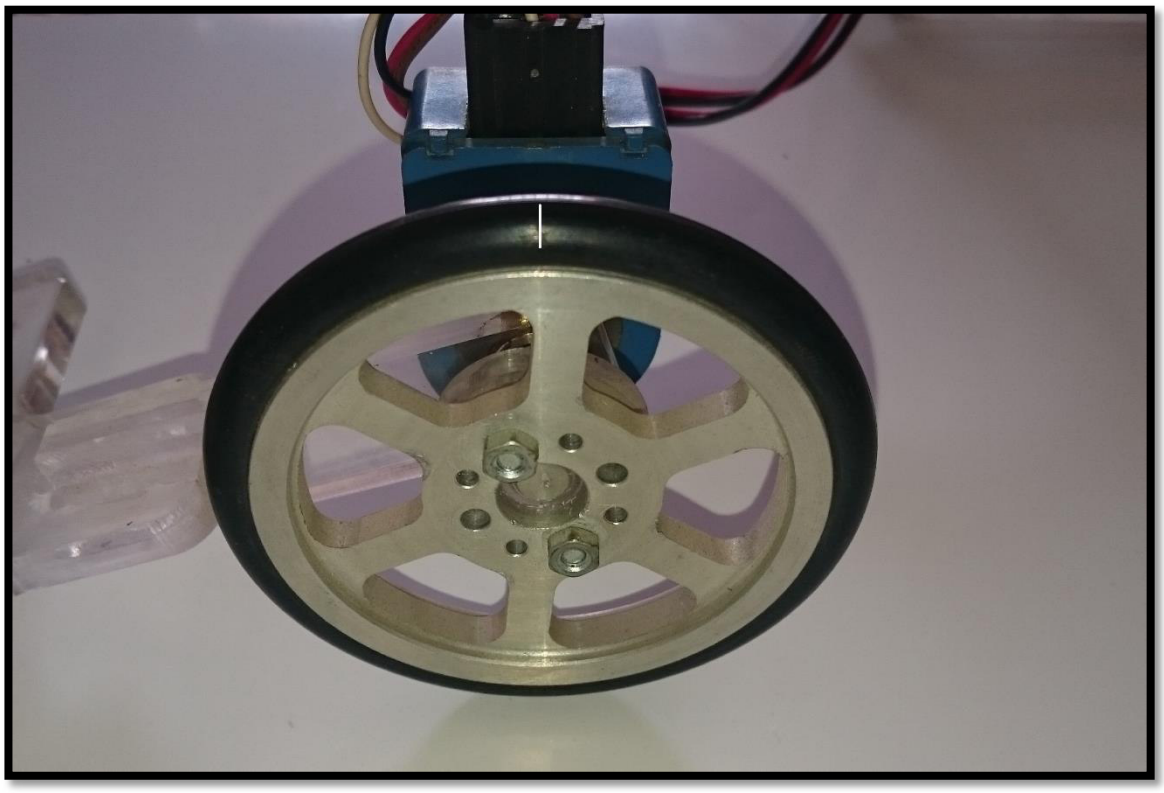


Figura 5.1.- Prueba 1.

Número de vueltas	Cantidad de ticks esperados	Cantidad de ticks contados
1/2 rev. (sentido horario)	250	250
1 rev. (sentido horario)	500	500
2 rev. (sentido horario)	1000	1000
5 rev. (sentido horario)	2500	2501
10 rev. (sentido horario)	5000	5001
20 rev. (sentido horario)	10000	10002
100 rev. (sentido horario)	50000	50005
1/2 rev. (sentido anti-horario)	- 250	- 250
1 rev. (sentido anti-horario)	- 500	- 500
2 rev. (sentido anti-horario)	- 1000	- 1000
5 rev. (sentido anti-horario)	- 2500	- 2499
10 rev. (sentido anti-horario)	- 5000	- 4998
20 rev. (sentido anti-horario)	- 10000	- 10003
100 rev. (sentido anti-horario)	- 50000	- 50004

Tabla 5.1.- Tabla de resultados de la prueba 1.

Prueba 2: Análisis odométrico

En esta prueba demuestra el correcto funcionamiento del algoritmo de análisis odométrico a través del cual se obtienen las coordenadas XY y la orientación angular del robot.

La prueba consiste en desplazar y girar el robot distancias y ángulos conocidos con el fin de verificar que la posición y orientación real correspondan con la obtenida de las ecuaciones del análisis odométrico (ver figura 5.2).

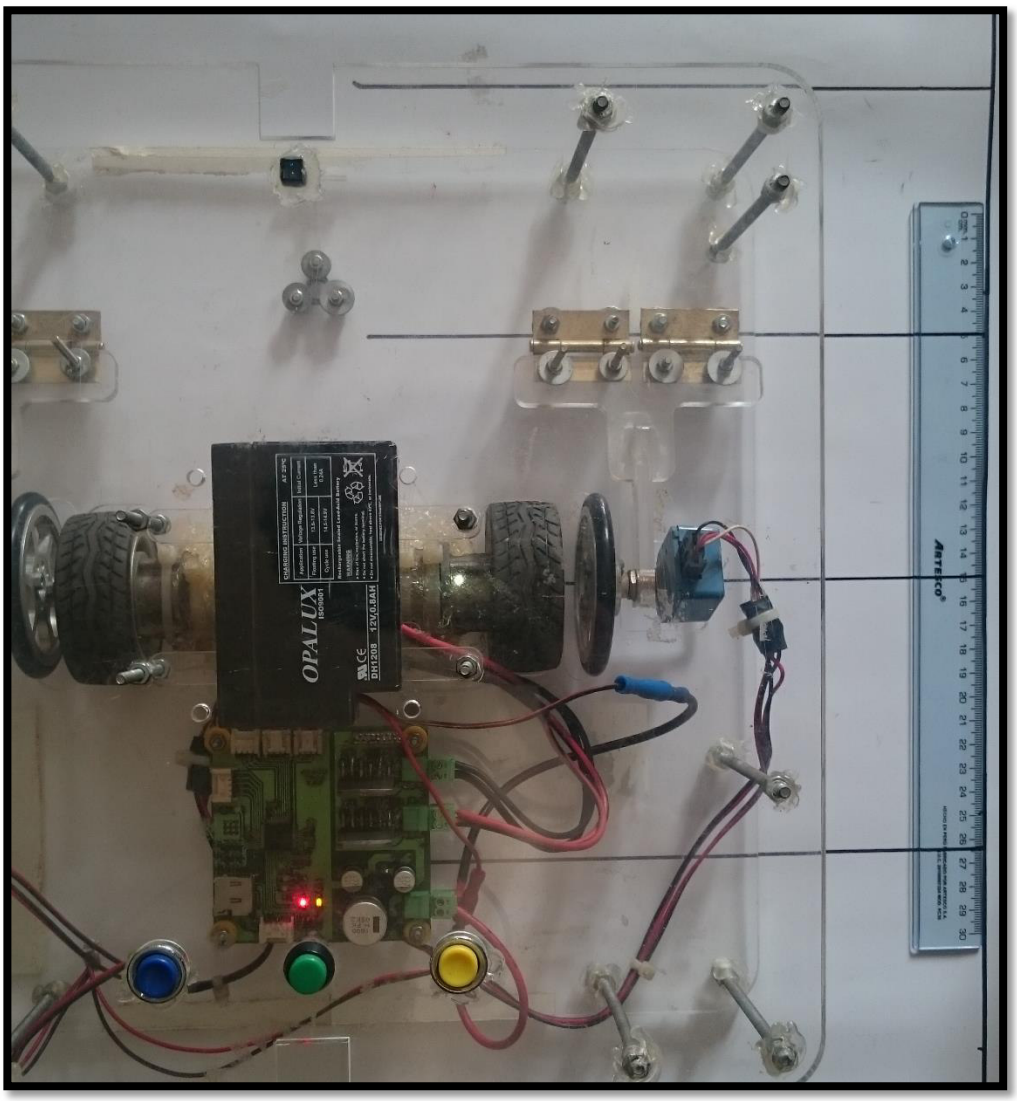


Figura 5.2.- Prueba 2.

Los resultados de las pruebas de medición de las coordenadas XY y orientación angular se presentan en las tablas 5.2 y 5.3 respectivamente, teniendo en cuenta que la posición inicial del robot es (0,0) con una orientación de 0º. En la primera tabla se observa la comparación entre la posición desplazada manualmente y a posición calculada a través de la odometría. Según estos resultados los valores calculados son muy cercanos a los reales con un error máximo de 0.08 cm. En la segunda tabla se muestra la comparación entre el ángulo rotado manualmente y el ángulo calculado. Al igual que el caso anterior existe un error muy pequeño.

Posición desplazada	Posición calculada
X = 0 cm , Y = 10 cm	X = 0.01cm , Y = 10.05 cm
X = 0 cm , Y = 20 cm	X = 0.01 cm , Y = 20.08 cm
X = 10 cm , Y = 0 cm	X = 10.02 cm , Y = 0.02 cm
X = 20 cm , Y = 0 cm	X = 20.02 cm , Y = 0.02 cm
X = 10 cm , Y = 10 cm	X = 10.03 cm , Y = 10.03 cm
X = 20 cm , Y = 30 cm	X = 20.03 cm , Y = 30.04 cm
X = 100 cm , Y = 100 cm	X = 100.08 cm , Y = 100.07 cm

Tabla 5.2.- Tabla de resultados de la prueba 2 - posición.

Ángulo rotado	Ángulo calculado
θ = 90º	θ = 90.01º
θ = 180º	θ = 180.03º
θ = 270º	θ = 270.03º
θ = 45º	θ = 45.0º
θ = 135º	θ = 135.04º
θ = 225º	θ = 225.05º
θ = 315º	θ = 315.05º

Tabla 5.3.- Tabla de resultados de la prueba 2 - ángulo.

Prueba 3: Control angular de 0º a 90º.

En esta prueba el robot se orientará de 0º a 90º, tal como se muestra en la figura 5.3, realizando un control angular. Se envió la orden desde la PC hacia el móvil en el cual se ejecutó la siguiente función para agregar la rutina a la secuencia.

$$add\_coordenadas(\quad 0,'N','N',pi/2 \quad)$$

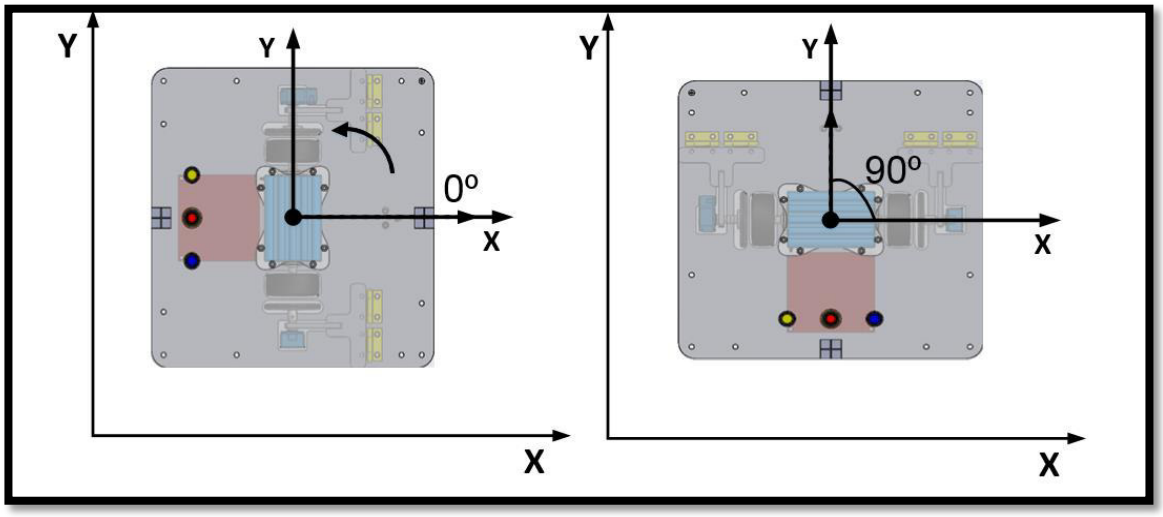


Figura 5.3.- Prueba 3.

La gráfica de la variación de la orientación angular obtenida en la prueba 3 se muestra en la figura 5.4.

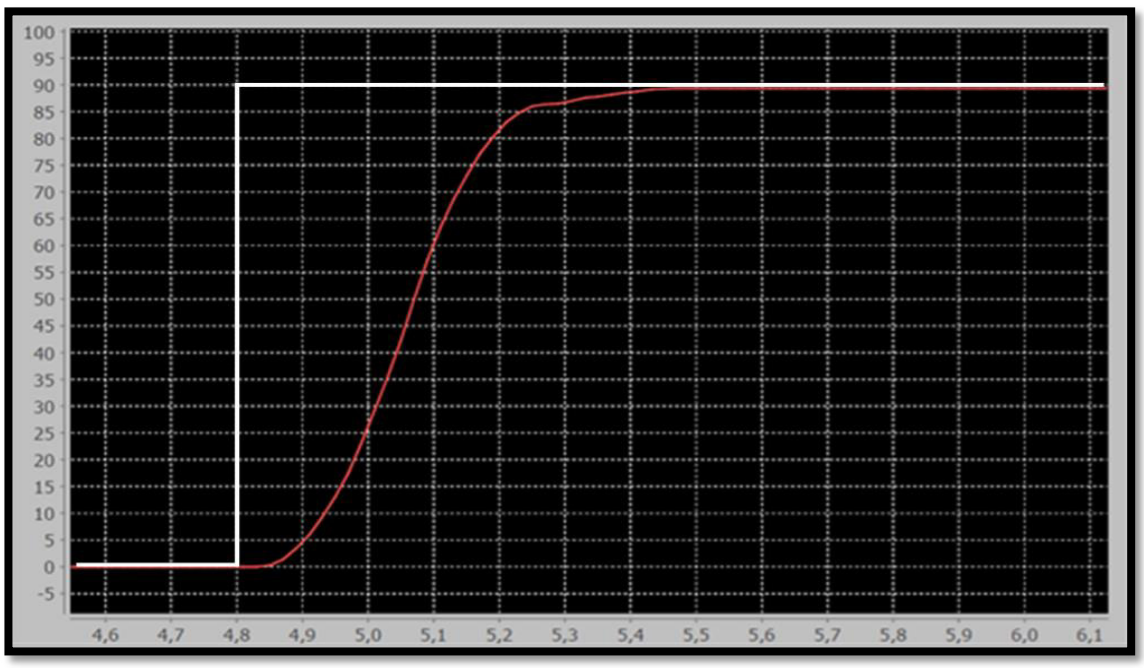


Figura 5.4.- Gráfica de la variación angular de la prueba 3.

Los resultados de la prueba se muestran en la tabla 5.4. Se observa que el error en estado estacionario (0.13º) es menor a la histéresis (1º), la cual se definió en el capítulo 3, 4.Desplazamiento. Así mismo, no se presentó sobre impulso durante el movimiento.

Parámetro	Valor
Ángulo inicial	0º
SP de ángulo	90º
Ángulo en estado estacionario	89.87º
Error en estado estacionario	0.13º
Tiempo de establecimiento	0.65 s
Sobre impulso	0%

Tabla 5.4.- Tabla de resultados de la prueba 3.

Prueba 4: Control angular de 0º a 45º

En esta prueba el robot se orientará de 0º a 45º, tal como se muestra en la figura 5.5, realizando un control angular. Se envió la orden desde la PC hacia el móvil en el cual se ejecutó la siguiente función para agregar la rutina a la secuencia.

$$add\_coordenadas(\quad 0,'N','N',pi/4\quad)$$

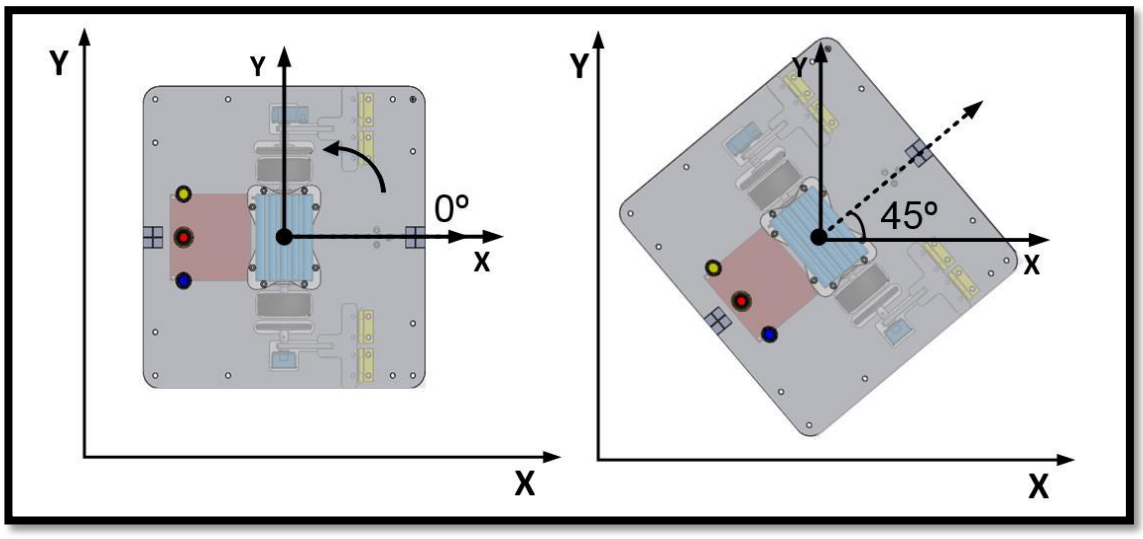


Figura 5.5.- Prueba 4.

La gráfica de la variación de la orientación angular obtenida en la prueba 4 se muestra en la figura 5.6.

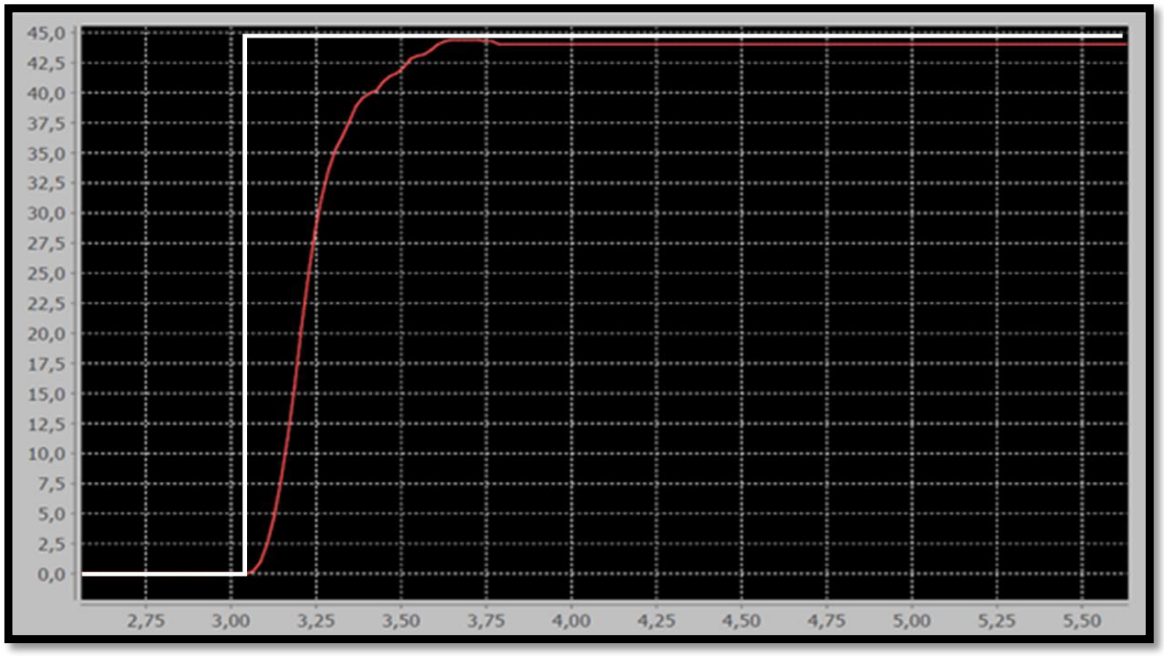


Figura 5.6.- Gráfica de la variación angular de la prueba 4.

Los resultados de la prueba se muestran en la tabla 5.5. Se observa que el error en estado estacionario (0.95º) es menor a la histéresis (1º). Así mismo, no se presentó sobre impulso durante el movimiento.

Parámetro	Valor
Ángulo inicial	0º
SP de ángulo	45º
Ángulo en estado estacionario	44.05º
Error en estado estacionario	0.95º
Tiempo de establecimiento	0.70 s
Sobre impulso	0%

Tabla 5.5.- Tabla de resultados de la prueba 4.

Prueba 5: Control angular de 0º a 180º

En esta prueba el robot se orientará de 0º a 180º, tal como se muestra en la figura 5.7, realizando un control angular. Se envió la orden desde la PC hacia el móvil en el cual se ejecutó la siguiente función para agregar la rutina a la secuencia.

$$add\_coordenadas(\quad 0,'N','N',pi\quad)$$

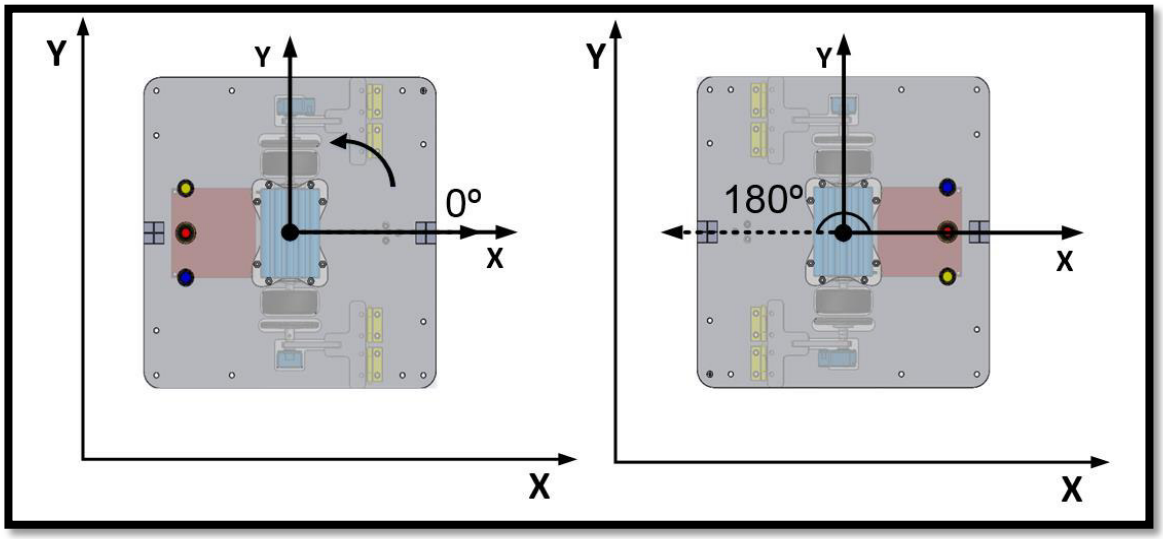


Figura 5.7.- Prueba 5.

La gráfica de la variación de la orientación angular obtenida en la prueba 5 se muestra en la figura 5.8.

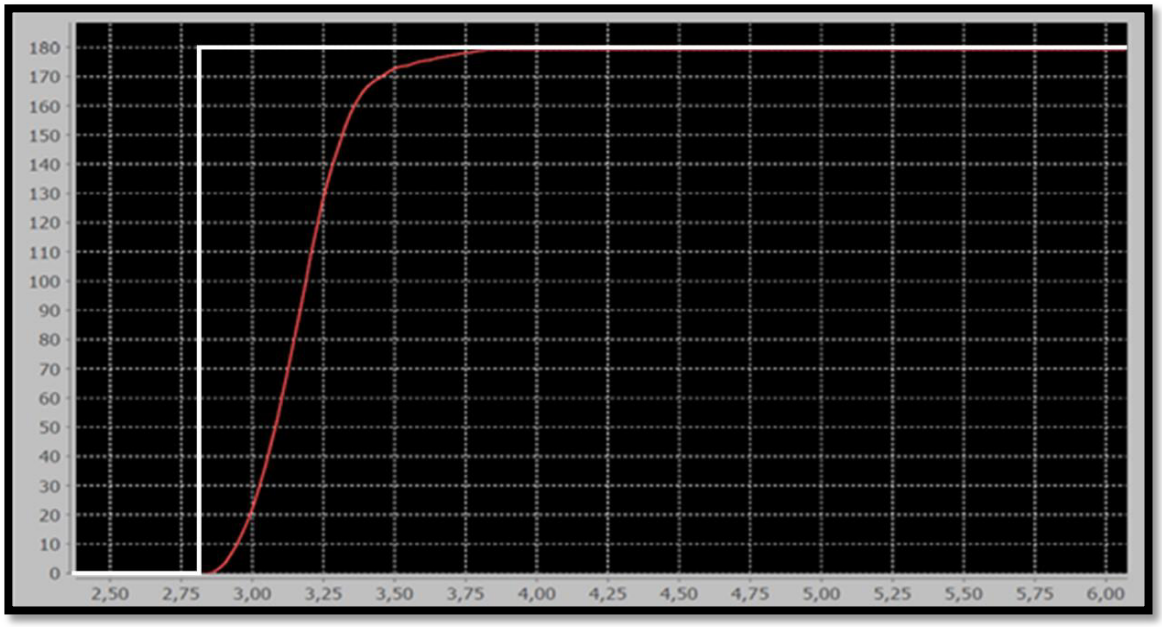


Figura 5.8.- Gráfica de la variación angular de la prueba 5.

Los resultados de la prueba se muestran en la tabla 5.6. Se observa que el error en estado estacionario (0.05º) es menor a la histéresis (1º). Así mismo, no se presentó sobre impulso durante el movimiento.

Parámetro	Valor
Ángulo inicial	0º
SP de ángulo	180º
Ángulo en estado estacionario	179.95º
Error en estado estacionario	0.05º
Tiempo de establecimiento	1.1 s
Sobre impulso	0%

Tabla 5.6.- Tabla de resultados de la prueba 5.

Prueba 6: Control angular de 90º a 45º

En esta prueba el robot se orientará de 90º a 45º, tal como se muestra en la figura 5.9, realizando un control angular. Se envió la orden desde la PC hacia el móvil en el cual se ejecutó la siguiente función para agregar la rutina a la secuencia.

$$add\_coordenadas(\quad 0,'N','N',pi/4\quad)$$

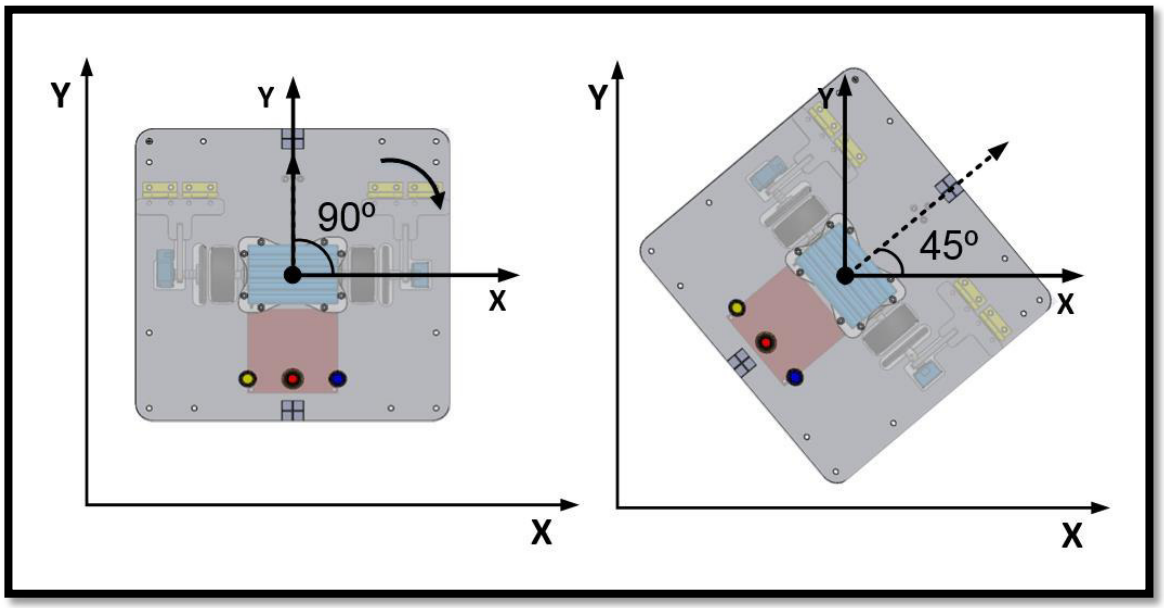


Figura 5.9.- Prueba 6.

La gráfica de la variación de la orientación angular obtenida en la prueba 6 se muestra en la figura 5.10.

Figura 5.10.- Gráfica de la variación angular de la prueba 6.

Los resultados de la prueba se muestran en la tabla 5.7. Se observa que el error en estado estacionario (0.12º) es menor a la histéresis (1º). Así mismo, se presentó sobre impulso de 0.88% el cual no produjo una variación considerable en el movimiento del robot.

Parámetro	Valor
Ángulo inicial	90º
SP de ángulo	45º
Ángulo en estado estacionario	45.12º
Error en estado estacionario	0.12º
Tiempo de establecimiento	0.9 s
Sobre impulso	0.88%

Tabla 5.7.- Tabla de resultados de la prueba 6.

Prueba 7: Control angular de 180º a 360º

En esta prueba el robot se orientará de 180º a 360º, tal como se muestra en la figura 5.11, realizando un control angular. Se envió la orden desde la PC hacia el móvil en el cual se ejecutó la siguiente función para agregar la rutina a la secuencia.

$$add\_coordenadas(\quad 0, 'N', 'N', 2pi \quad)$$

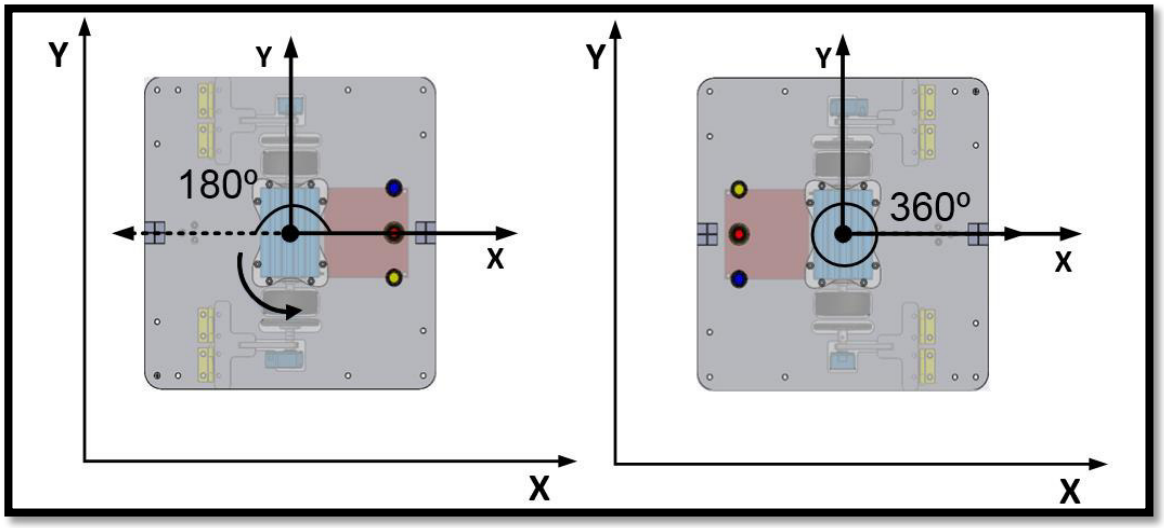


Figura 5.11.- Prueba 7.

La gráfica de la variación de la orientación angular obtenida en la prueba 7 se muestra en la figura 5.12.

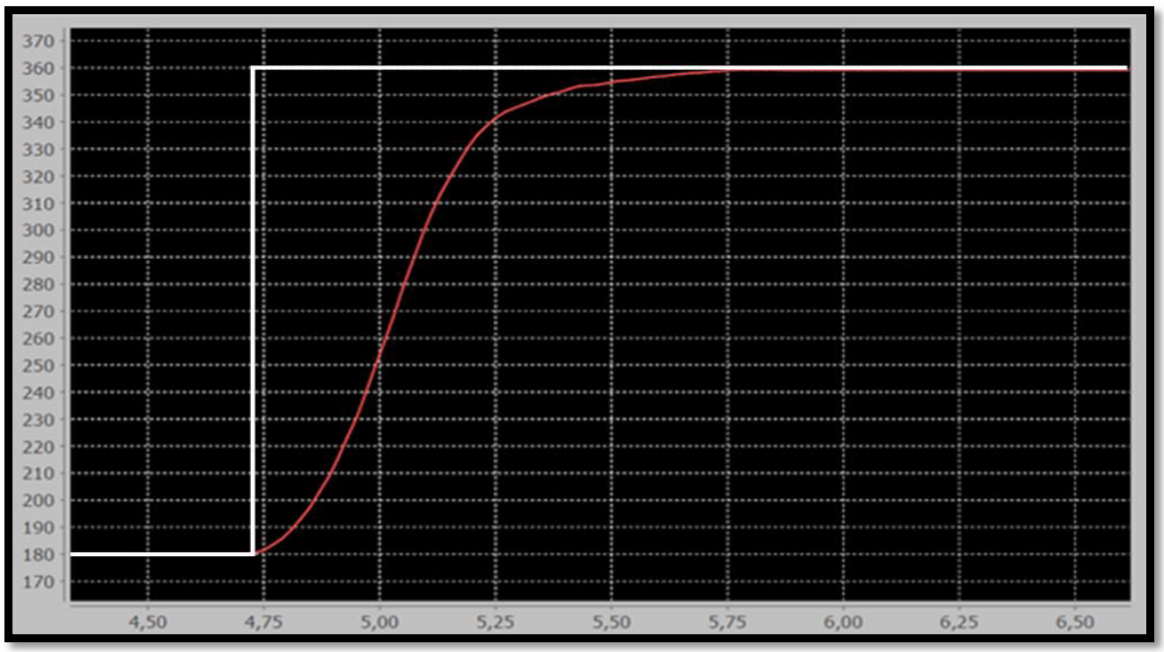


Figura 5.12.- Gráfica de la variación angular de la prueba 7.

Los resultados de la prueba se muestran en la tabla 5.8. Se observa que el error en estado estacionario (0.04º) es menor a la histéresis (1º). Así mismo, no se presentó sobre impulso durante el movimiento.

Parámetro	Valor
Ángulo inicial	180º
SP de ángulo	360º
Ángulo en estado estacionario	359.96º
Error en estado estacionario	0.04º
Tiempo de establecimiento	1.00 s
Sobre impulso	0%

Tabla 5.8.- Tabla de resultados de la prueba 7.

Prueba 8: Control angular de 270º a 45º

En esta prueba el robot se orientará de 270º a 45º, tal como se muestra en la figura 5.13, realizando un control angular. Se envió la orden desde la PC hacia el móvil en el cual se ejecutó la siguiente función para agregar la rutina a la secuencia.

$$add\_coordenadas(\quad 0,'N','N',pi/4\quad)$$

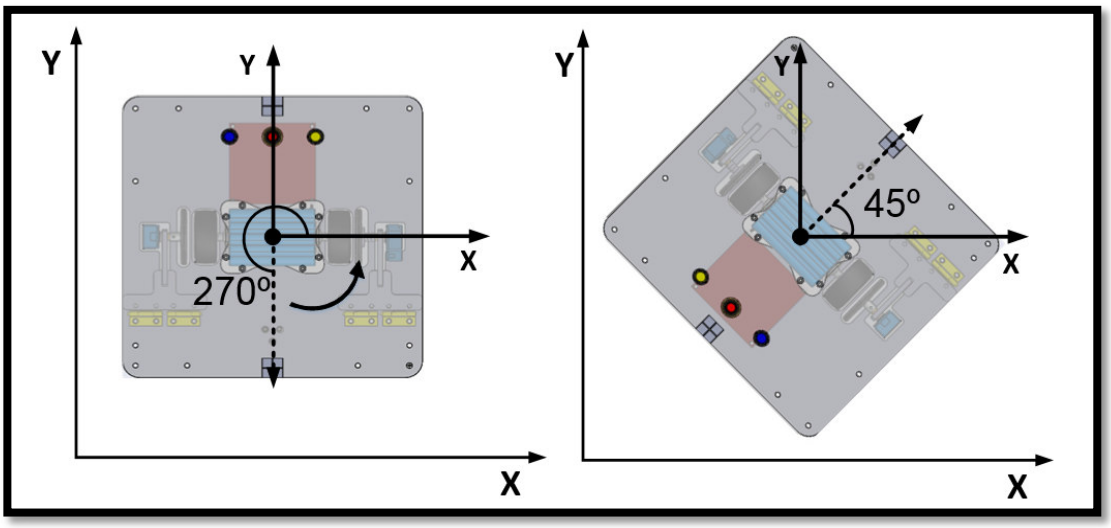


Figura 5.13.- Prueba 7.

La gráfica de la variación de la orientación angular obtenida en la prueba 8 se muestra en la figura 5.14.

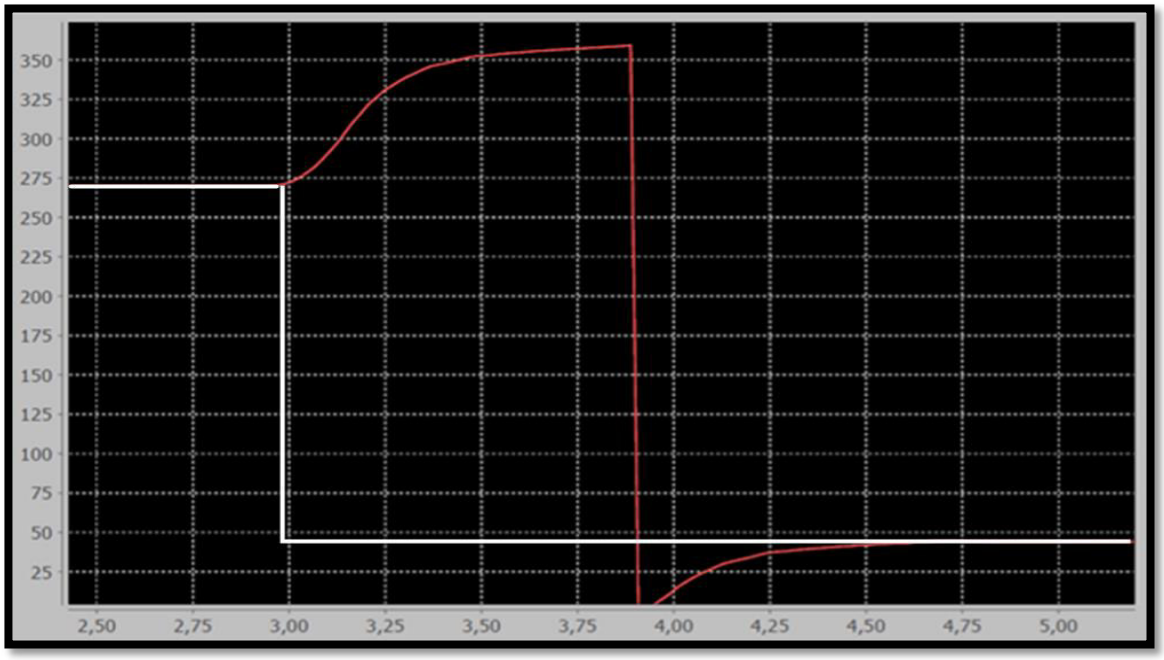


Figura 5.14.- Gráfica de la variación angular de la prueba 8.

En la gráfica anterior se observó el funcionamiento del algoritmo de la función "control_ang_rot" donde se busca que el robot recorra la menor distancia angular al momento de hacer el control angular. En este caso se realiza un control de 270º a 45º por lo que hay dos caminos:

- En sentido horario : 270º - 45º = 225º
- En sentido anti horario : 360º - 270º + 45º = 135º

El algoritmo funcionó correctamente ya que el robot giró en sentido anti horario, recorriendo la menor distancia angular.

Los resultados de la prueba se muestran en la tabla 5.9. Se observa que el error en estado estacionario (0.02º) es menor a la histéresis (1º). Así mismo, no se presentó sobre impulso durante el movimiento.

Parámetro	Valor
Ángulo inicial	270º
SP de ángulo	45º
Ángulo en estado estacionario	44.98º
Error en estado estacionario	0.02º
Tiempo de establecimiento	1.50 s
Sobre impulso	0%

Tabla 5.9.- Tabla de resultados de la prueba 8.

Prueba 9: Control angular de 90º a 0º, 90º, 45º, 135º y 0º.

En esta prueba se realizará la orientación a una secuencia de ángulos (ver figura 5.15) para comprobar que el robot no pierda la orientación al girar en diferentes sentidos. Las secuencias fueron agregadas a través de las siguientes funciones.

$$add\_coordenadas(\ 0,'N','N',0\ )$$
$$add\_coordenadas(\ 0,'N','N',pi/2\ )$$
$$add\_coordenadas(\ 0,'N','N',pi/4\ )$$
$$add\_coordenadas(\ 0,'N','N',3\,pi/4\ )$$
$$add\_coordenadas(\ 0,'N','N',0\ )$$

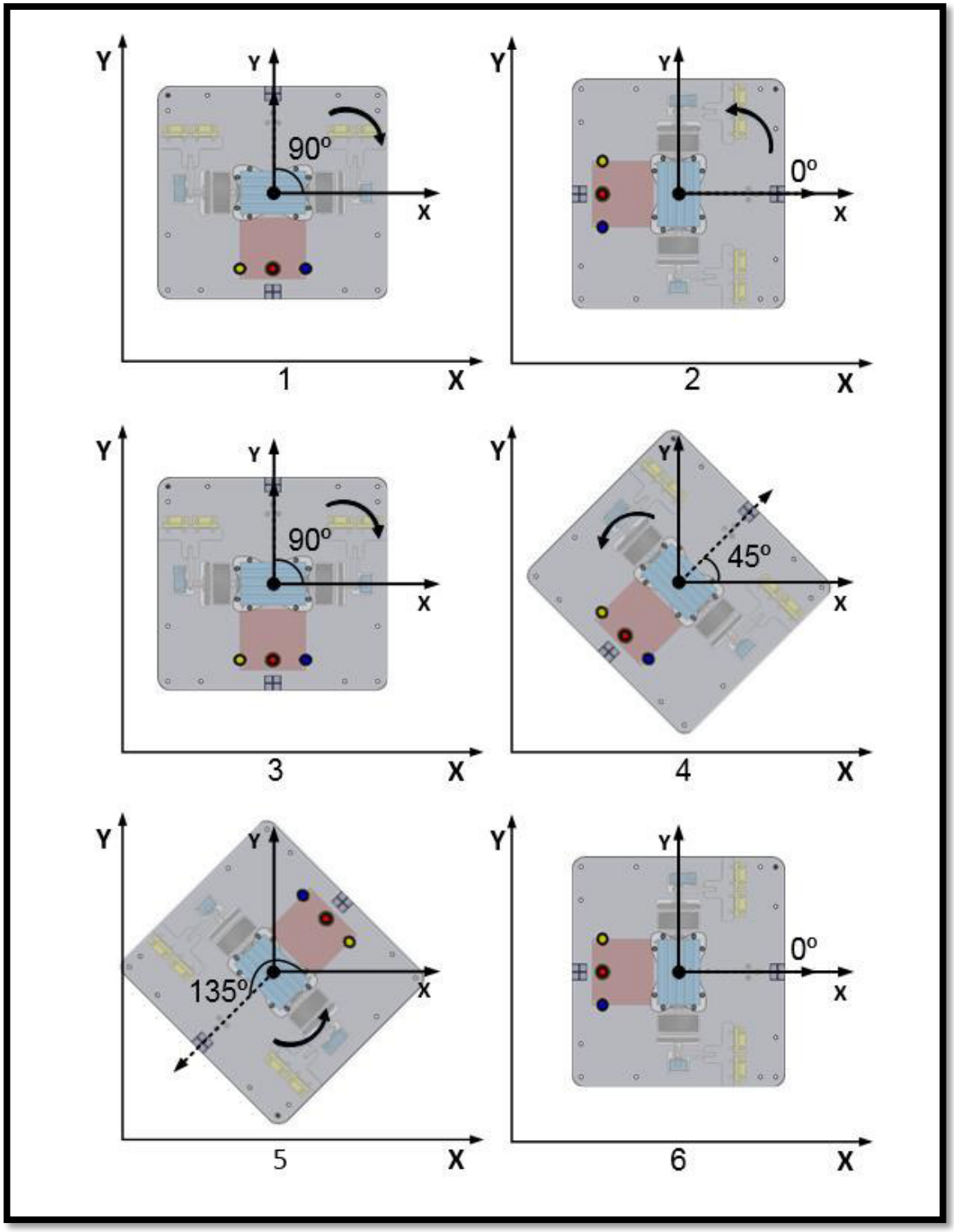


Figura 5.15.- Prueba 9.

La gráfica de la variación de la orientación angular obtenida en la prueba 9 se muestra en la figura 5.16.

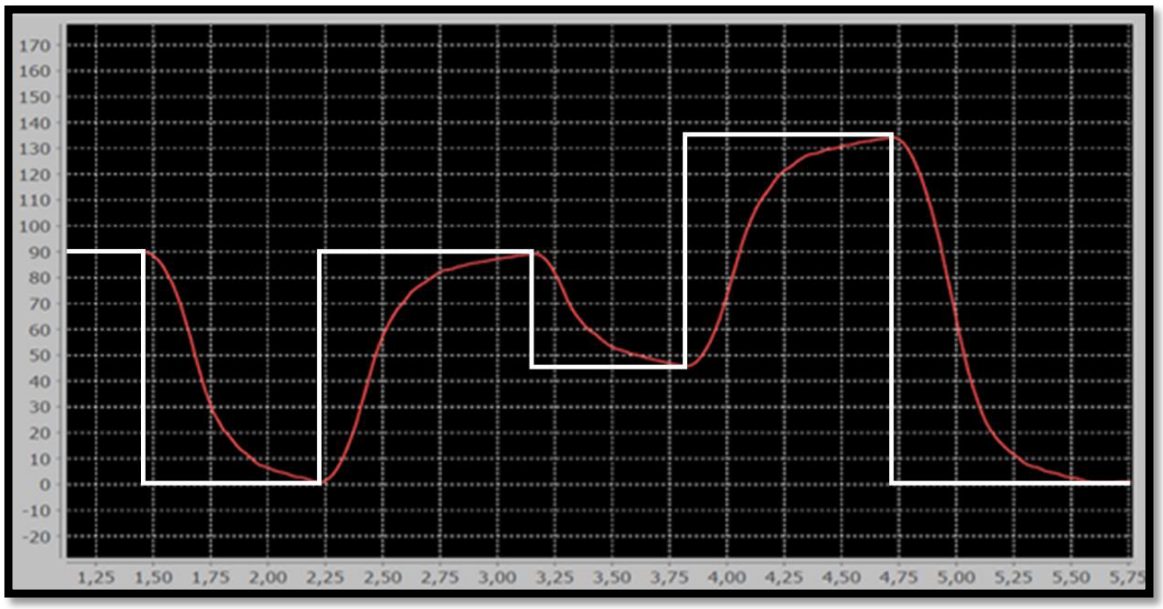


Figura 5.16.- Gráfica de la variación angular de la prueba 9.

Los resultados de la prueba se muestran en la tabla 5.10. Donde en todos los puntos se obtuvo en error inferior a la histéresis. Además no se presentaron sobre impulsos durante movimiento. Así mismo se cumplió la secuencia en el orden que se agregaron las columnas con la función “add_coordenadas”.

Parámetro	Valor
(1) Ángulo inicial	90º
(2) SP de ángulo	0º
(2) Ángulo en estado estacionario	0.10º
(2) Error en estado estacionario	0.10º
(2) Tiempo de establecimiento	0.75 s
(2) Sobre impulso	0%
(3) SP de ángulo	90º
(3) Ángulo en estado estacionario	89.80º
(3) Error en estado estacionario	0.20º
(3) Tiempo de establecimiento	0.90 s
(3) Sobre impulso	0%

(4) SP de ángulo	45º
(4) Ángulo en estado estacionario	44.94º
(4) Error en estado estacionario	0.06º
(4) Tiempo de establecimiento	0.71 s
(4) Sobre impulso	0%
(5) SP de ángulo	135º
(5) Ángulo en estado estacionario	134.97º
(5) Error en estado estacionario	0.03º
(5) Tiempo de establecimiento	0.85 s
(5) Sobre impulso	0%
(6) SP de ángulo	0º
(6) Ángulo en estado estacionario	0.08º
(6) Error en estado estacionario	0.08º
(6) Tiempo de establecimiento	0.75 s
(6) Sobre impulso	0%

Tabla 5.10.- Tabla de resultados de la prueba 9.

Prueba 10: Control de posición de (20,20) a (20,30)

En esta prueba el robot se desplazará del punto P1 (20,20) al punto P2 (20,30), tal como se muestra en la figura 5.17, realizando un control posición. Además, el robot inicialmente posee una orientación angular de 90º. Se envió la orden desde la PC hacia el móvil en el cual se ejecutó la siguiente función para agregar la rutina a la secuencia.

$$add\_coordenadas(\ 1,\ \ 20,\ \ 30,\ \ 'N'\ )$$

La gráfica de la variación de la posición del robot obtenida en la prueba 8 se muestra en la figura 5.18.

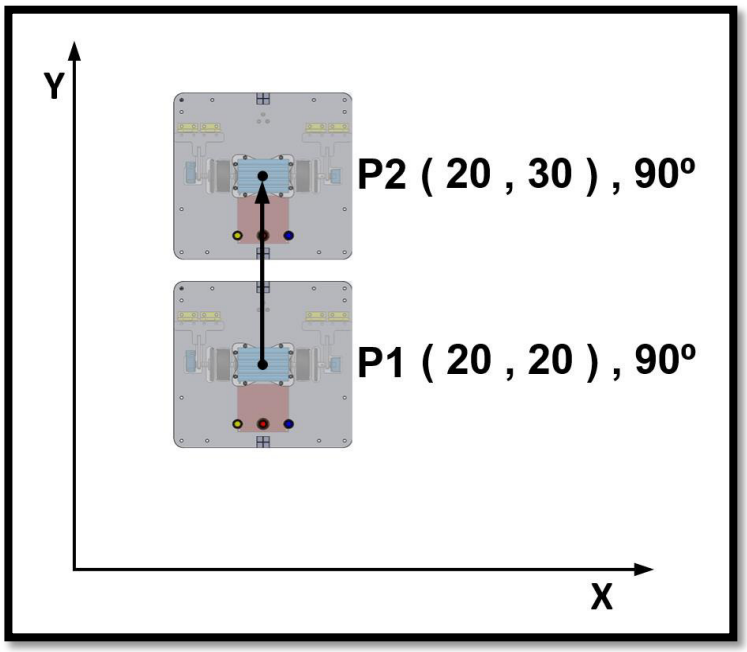


Figura 5.17.- Prueba 10.

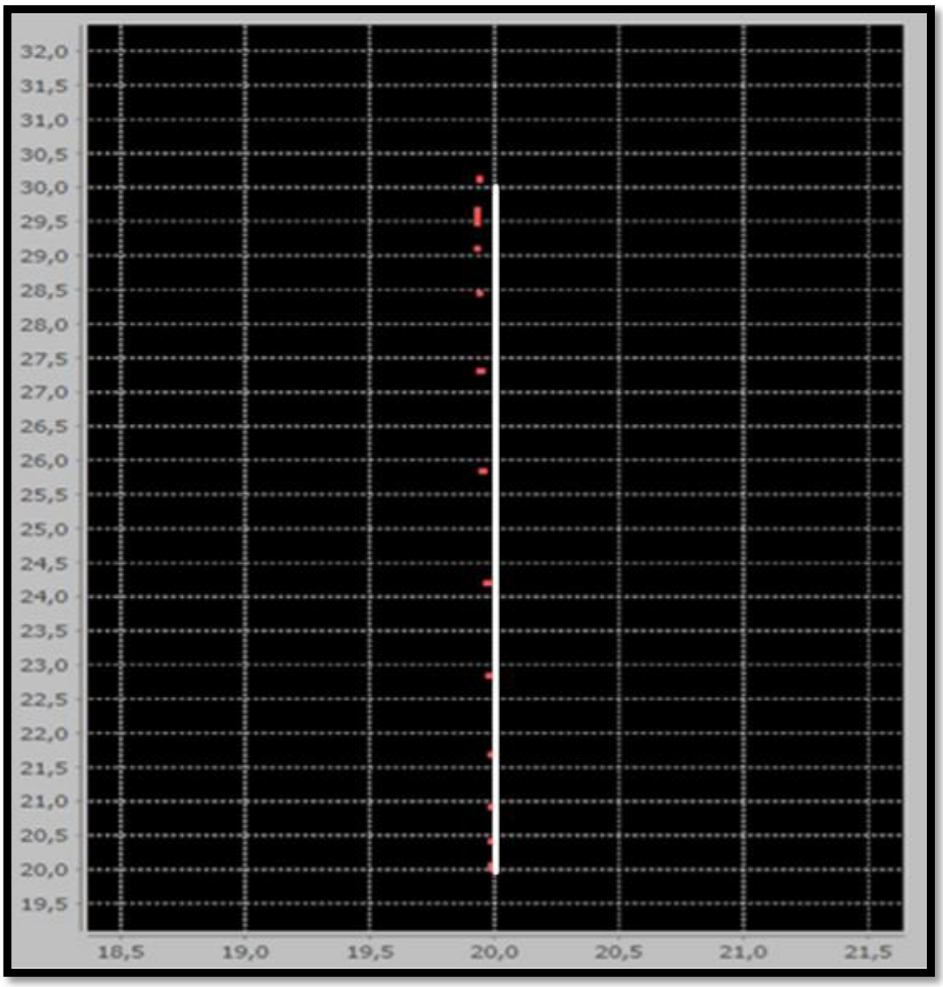


Figura 5.18.- Grafica de la variación de posición (x,y) de la prueba 10.

Los resultados de la prueba se muestran en la tabla 5.11. El robot se desplaza en línea recta manteniendo su orientación de 90º, el error registrado al culminar su recorrido es de 0.15 cm.

Parámetro	Valor
Ángulo inicial	90º
Posición inicial (x,y)	(20, 20)
SP de posición	(20, 30)
SP de ángulo	90º
Posición en estado estacionario	(19.95 , 30.15)
Error de posición en estado estacionario	0.15 cm

Tabla 5.11.- Tabla de resultados de la prueba 10.

Prueba 11: Control de posición de (20,20) a (30,30)

En esta prueba el robot se desplazará del punto P1 (20,20) al punto P2 (30,30), tal como se muestra en la figura 5.19, realizando un control posición. Además, el robot inicialmente posee una orientación angular de 90º. Se envió la orden desde la PC hacia el móvil en el cual se ejecutó la siguiente función para agregar la rutina a la secuencia.

$$add\_coordenadas(\ 1,\ \ 30,\ \ 30,\ \ 'N'\ )$$

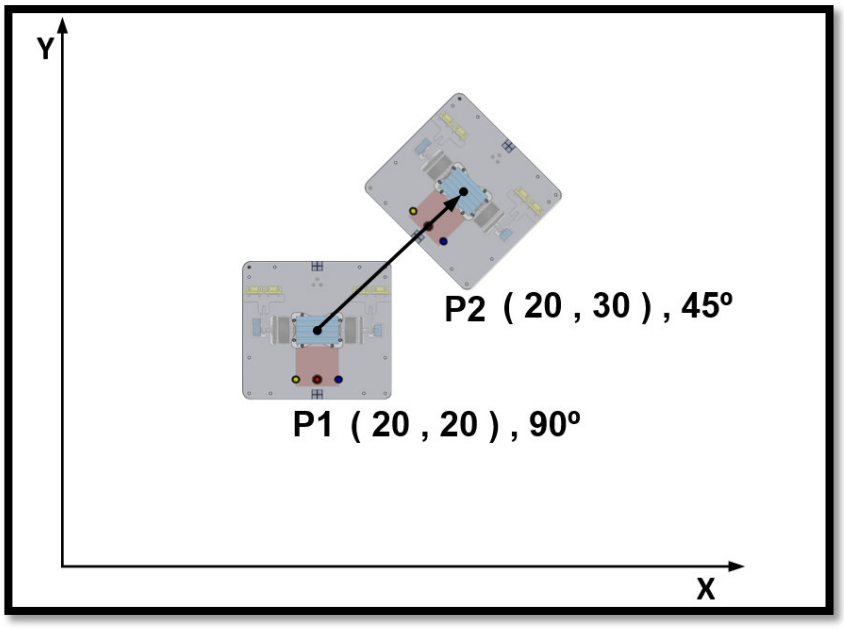


Figura 5.19.- Prueba 11.

La gráfica de la variación de la orientación angular del robot obtenida en la prueba 11 se muestra en la figura 5.20.

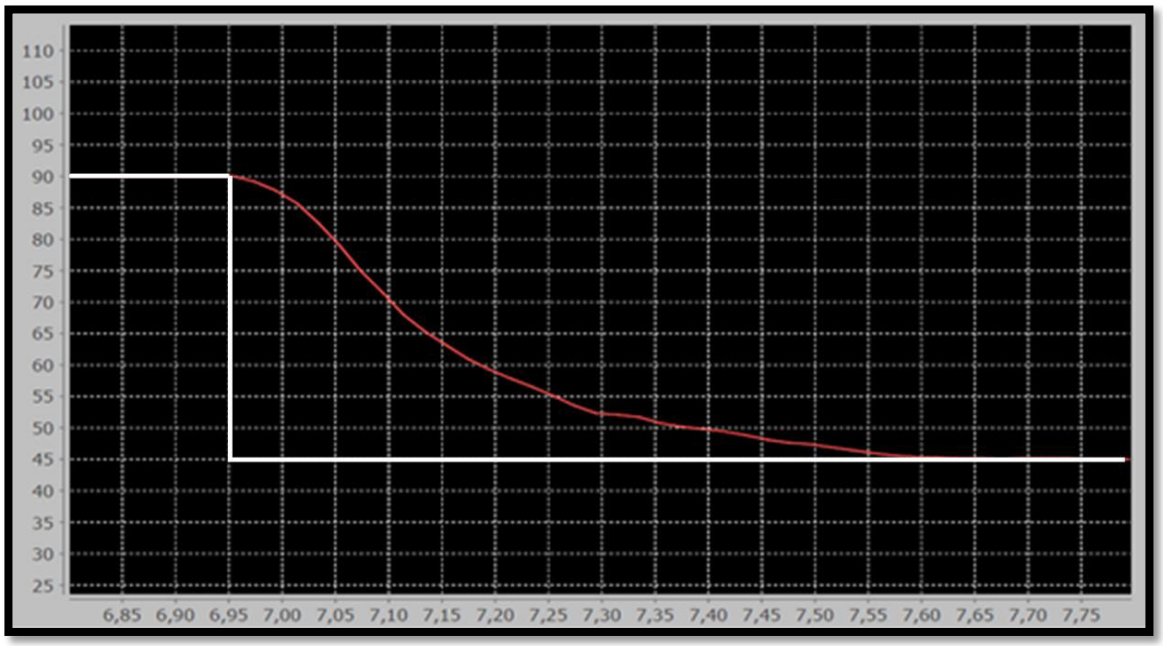


Figura 5.20.- Gráfica de la variación angular de la prueba 11.

La gráfica de la variación de la posición del robot se muestra en la figura 5.21.

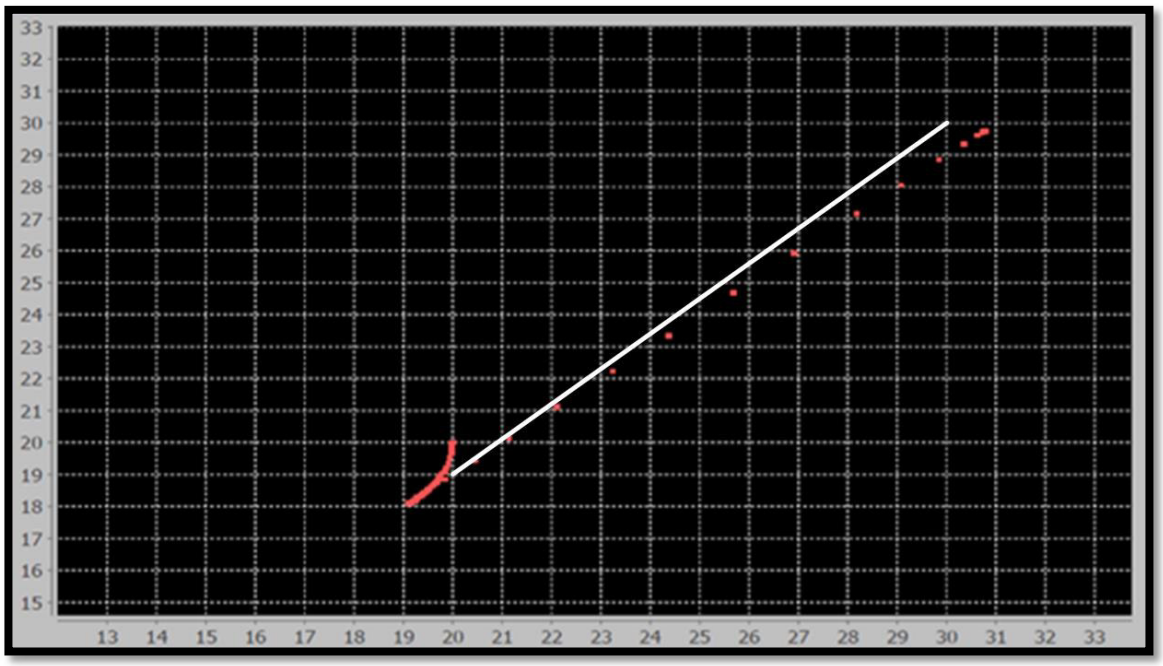


Figura 5.21.- Grafica de la variación de posición (x,y) de la prueba 11.

En la prueba 11 se comprueba el algoritmo de la función "control_trayectoria", en la cual primero se realiza un control de orientación hacia el punto que se dirigirá el robot, en este caso 45º . Luego se ejecuta el control de posición, generando un error final de 0.80 cm. Los resultados de la prueba se muestran en la tabla 5.12

Parámetro	Valor
Ángulo inicial	90º
Posición inicial (x,y)	(20, 20)
SP de posición	(30, 30)
SP de ángulo	45º
Ángulo en estado estacionario	44.20º
Error de ángulo en estado estacionario	0.80º
Posición en estado estacionario	(30.80 , 29.90)
Error de posición en estado estacionario	0.80 cm

Tabla 5.12.- Tabla de resultados de la prueba 11.

Prueba 12: Control de posición de (20,20) a (20,30) y (30,30)

En esta prueba el robot se desplazará del punto P1 (20,20) al punto P2 (20,30) y P3 (30.30) tal como se muestra en la figura 5.22, realizando un control posición. Además, el robot inicialmente posee una orientación angular de 90º. Se envió la orden desde la PC hacia el móvil en el cual se ejecutó la siguiente función para agregar la rutina a la secuencia.

$$add\_coordenadas(\ 1,\ 20,\ 30,\ 'N'\ )$$
$$add\_coordenadas(\ 1,\ 30,\ 30,\ 'N'\ )$$

La gráfica de la variación de la orientación angular del robot obtenida en la prueba 12 se muestra en la figura 5.23.

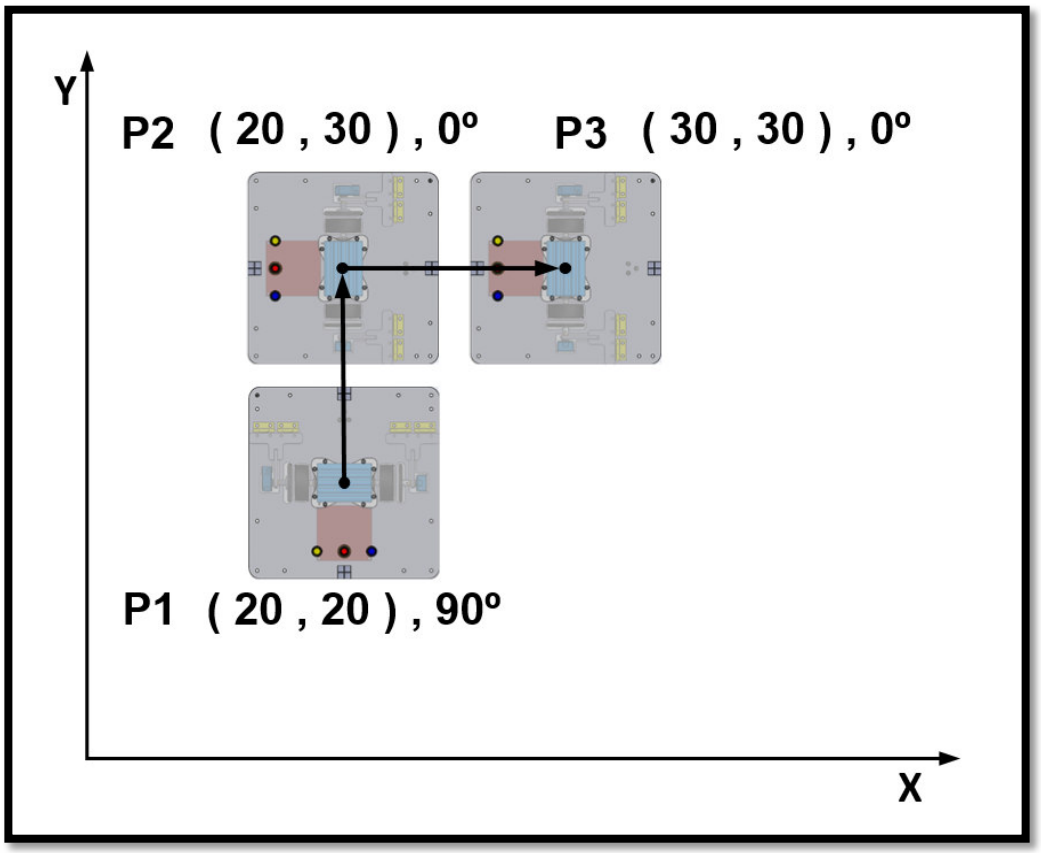


Figura 5.22.- Prueba 12.

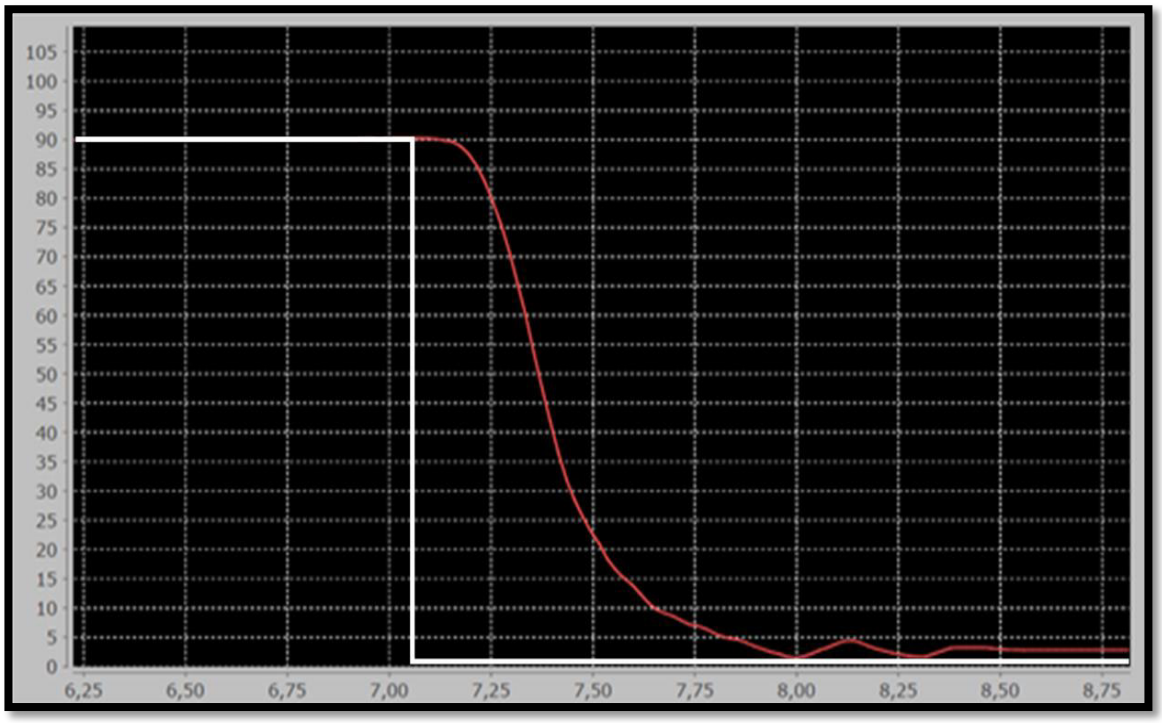


Figura 5.23.- Gráfica de la variación angular de la prueba 12.

La gráfica de la variación de la posición del robot se muestra en la figura 5.24.

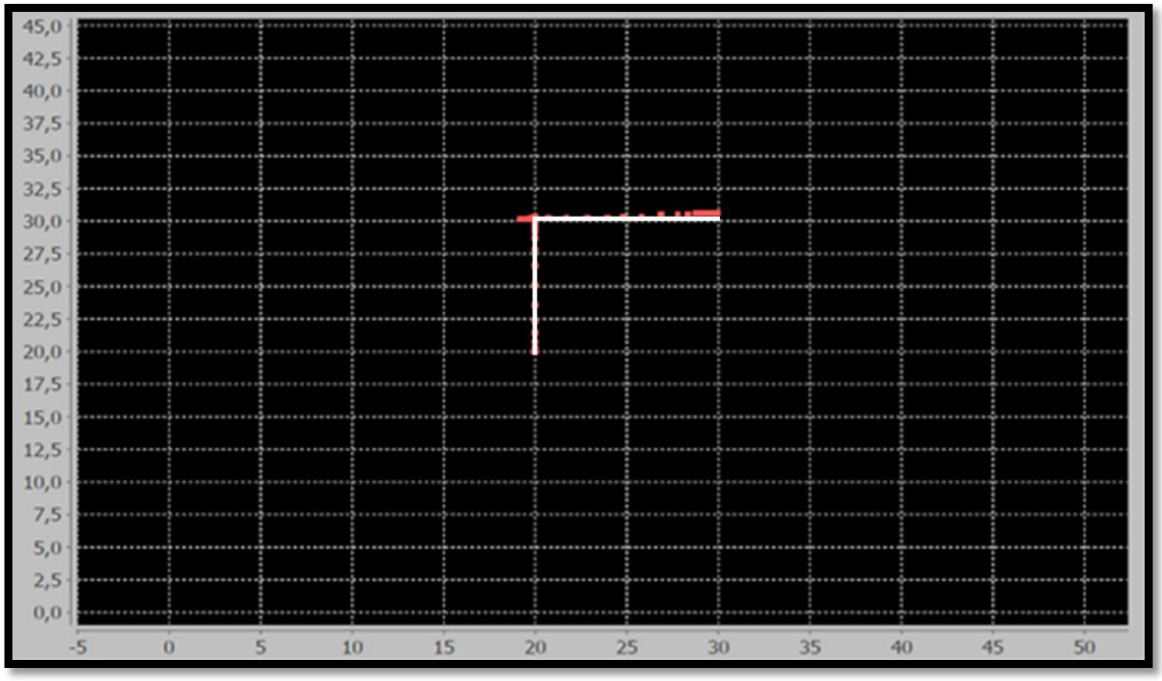


Figura 5.24.- Gráfica de la variación de posición de la prueba 12.

En la prueba 12 se realiza un desplazamiento lineal de 10 cm seguido de un cambio de orientación de 90 a 0 grados y nuevamente un recorrido de 10cm. Se puede observar que el robot ejecuta correctamente el control de posición y orientación, según la tabla 5.13 los errores son inferiores a las histéresis de ángulo (1º) y de posición (1cm).

Parámetro	Valor
(P1) Ángulo inicial	90º
(P1) Posición inicial (x,y)	(20, 20)
(P1 – P2) SP de posición	(20, 30)
(P1 – P2) SP de ángulo	90º
(P1 – P2) Ángulo en estado estacionario	89.96º
(P1 – P2) Error de ángulo en estado estacionario	0.04º
(P1 – P2) Posición en estado estacionario	(20.05 , 30.02)
(P1 – P2) Error de posición en estado estacionario	0.05 cm

(P2 – P3) SP de posición	(30, 30)
(P2 – P3) SP de ángulo	0º
(P2 – P3) Ángulo en estado estacionario	0.96º
(P2 – P3) Error de ángulo en estado estacionario	0.96º
(P2 – P3) Posición en estado estacionario	(30.10 , 30.30)
(P2 – P3) Error de posición en estado estacionario	0.32 cm

Tabla 5.13.- Tabla de resultados de la prueba 12.

Prueba 13: Control de posición de (20,20) a (30,30), (40,20) y (20,20)

En esta prueba el robot se desplazará del punto P1 (20,20) al punto P2 (30,30), P3 (30.30) y nuevamente al P1 (20,20), tal como se muestra en la figura 5.25, realizando un control posición. Además, el robot inicialmente posee una orientación angular de 90º. Se envió la orden desde la PC hacia el móvil en el cual se ejecutó la siguiente función para agregar la rutina a la secuencia.

$$add\_coordenadas(\ 1,\ 30,\ 30,\ 'N'\ )$$
$$add\_coordenadas(\ 1,\ 40,\ 20,\ 'N'\ )$$
$$add\_coordenadas(\ 1,\ 20,\ 20,\ 'N'\ )$$

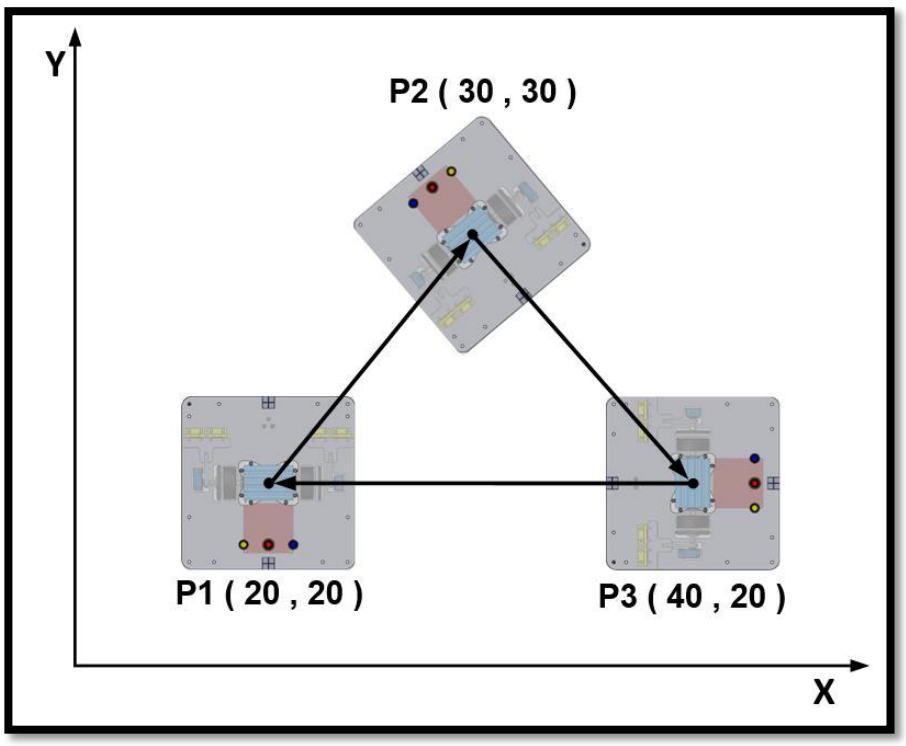


Figura 5.25.- Prueba 13.

EL robot deberá cambiar su orientación a 45º para dirigirse del punto P1 al punto P2. Al llegar a este punto tendrá cambiar si orientación a 315º para desplazarse al punto P3. Finalmente, el robot se orientará a 180º para regresar al punto P1.

La gráfica de la variación de la orientación angular del robot obtenida en la prueba 13 se muestra en la figura 5.26.

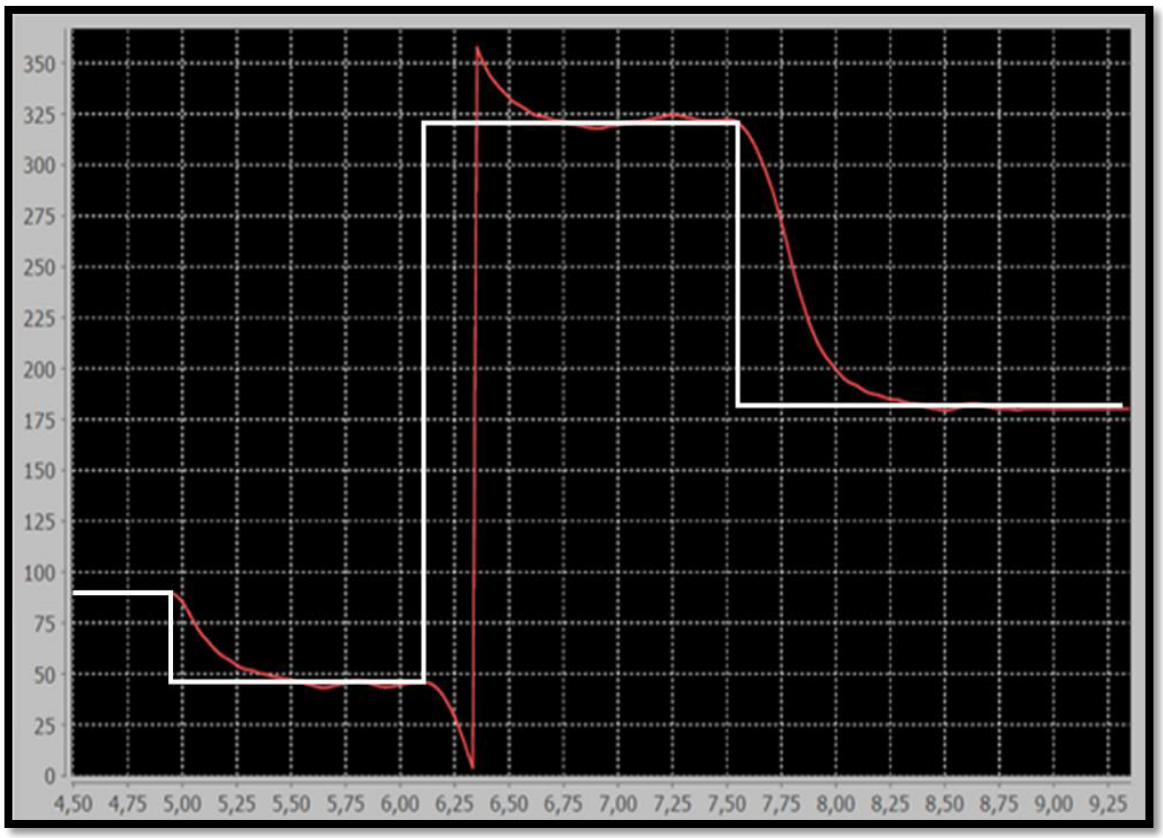


Figura 5.26.- Gráfica de la variación angular de la prueba 13.

En esta prueba se observa el control de la distancia a la recta, según la gráfica de la figura 5.27, la posición del robot se desplaza cuando el robot se orienta a 315º para dirigirse al punto P3. Este desplazamiento produce que la distancia del centro del robot a la recta entre P2 y P3 sea mayor a 1cm, por lo que el robot se reorientará nuevamente para llegar al punto P3.

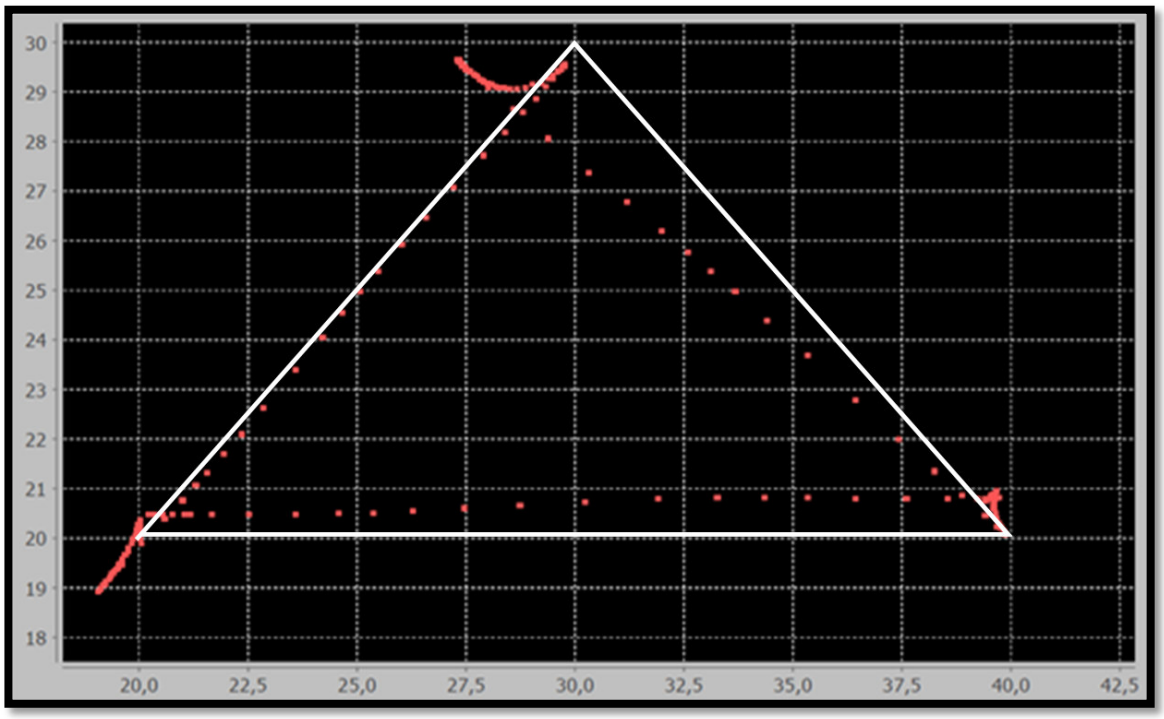


Figura 5.27.- Gráfica de la variación angular de la prueba 13.

Los resultados de la prueba se muestran en la tabla 5.14.

Parámetro	Valor
(P1) Ángulo inicial	90º
(P1) Posición inicial (x,y)	(20, 20)
(P1 – P2) SP de posición	(30, 30)
(P1 – P2) SP de ángulo	45º
(P1 – P2) Ángulo en estado estacionario	44.80º
(P1 – P2) Error de ángulo en estado estacionario	0.20º
(P1 – P2) Posición en estado estacionario	(29.86 , 29.65)
(P1 – P2) Error de posición en estado estacionario	0.38 cm
(P2 – P3) SP de posición	(40, 20)
(P2 – P3) SP de ángulo	320º
(P2 – P3) Ángulo en estado estacionario	320.30º
(P2 – P3) Error de ángulo en estado estacionario	0.30º
(P2 – P3) Posición en estado estacionario	(39.95 , 20.10)
(P2 – P3) Error de posición en estado estacionario	0.11 cm

(P3 – P1) SP de posición	(20, 20)
(P3 – P1) SP de ángulo	180º
(P3 – P1) Ángulo en estado estacionario	179.89º
(P3 – P1) Error de ángulo en estado estacionario	0.11º
(P3 – P1) Posición en estado estacionario	(20.10 , 20.40)
(P3 – P1) Error de posición en estado estacionario	0.41 cm

Tabla 5.14.- Tabla de resultados de la prueba 13.

Capítulo VI
Conclusiones

El algoritmo implementado para la lectura de encoders presentó buenos resultados. Según la tabla 5.1 obtenida en la prueba 1 del capítulo 5, la cantidad de ticks contabilizados por el algoritmo presentan un error mínimo en comparación a la cantidad de ticks esperados al momento de girar el eje del encoder. Este error es de aproximadamente 5 ticks por cada 100 revoluciones, donde cada revolución posee 500 ticks. En base a estos resultados se corroboró el correcto funcionamiento del algoritmo.

De acuerdo a las tablas 5.2 y 5.3 mostradas en la prueba 2 del capítulo 5, se comprobó el buen desempeño del análisis odométrico como método de obtención de las coordenadas XY y la orientación angular de robot. En estos resultados la precisión de los parámetros calculados a través de las ecuaciones de odometría fue muy cercana a los valores reales, presentando un error máximo de 0.08 cm en el cálculo de las coordenadas; y 0.05º en la orientación angular del robot.

El uso de PID's para la regulación de la posición y la orientación angular del robot permitió que las variables sean controladas de forma precisa. Según las pruebas realizadas en el capítulo anterior, ambos controladores funcionaron correctamente generando errores muy pequeños e inferiores a las histéresis de cada uno. Además, se corroboró la adecuada ejecución de la secuencia de puntos en los cuales debía posicionarse el robot. Esto demostró la gran utilidad de la función que agrega coordenadas en línea cuando el sistema se encuentra en funcionamiento. De igual forma se verificó el control de la distancia a la recta, el cual aseguró que el robot no se desvíe de la recta trazada entre los puntos que se desplaza, tal como observó en la prueba 13.

El uso de una interfaz gráfica fue determinante durante la fase de pruebas y sintonización de los controladores PID, así como para el monitoreo continuo durante las operaciones de desplazamiento del robot en tiempo real. Otra de las ventajas del empleo de una interfaz fue la visualización de gráficas de los distintos parámetros de control como la orientación angular, coordenadas XY, velocidad y nivel de batería.

Finalmente se pudo demostrar que es posible realizar un control eficiente y preciso de un robot móvil empleando únicamente dos encoders como sistema de adquisición y utilizando un sistema de control embebido dentro de un microcontrolador de alto rendimiento.

Referencias Bibliográficas

[1] Katsuhiko Ogata, Ingeniería de control moderna, vol. I México, 1998, p. 669.

[2] Microchip. "Datasheet dsPIC30F4011", 2005.

[3] Agre, Philip E. y Chapman David. " What are plans for?", Robotics and Autonomous Systems. No. 6 (1990); p. 17 – 34.

[4] IEEE Robotics and Automation Soviet 2001.

[5] Alfonso Romero, "Control y Robótica", Sección Encoders, Abril 2008.

[6] Feng, L, Borenstein, J y Everett, H.R. "Where am I?" Sensors and methods for mobile robot positioning Universidad de Michigan. 1996.

[7] Danilo Navarro, Gines Benet Gilabert, Luís Hernando Ríos, Maximiliano Bueno "Mejoras de la localización odométrica de un robot diferencial mediante la corrección de errores sistemáticos". Revista Scientia et Técnica vol XIII número 37.

[8] E. Eitelberg, "A regulating and tracking PID controller", 1987.

[9] D. Powell, "Digital control of dinamic systems", 1980.

[10] P. Gawthrop, "Self tuning PI and PID controllers", IEEE conference on aplications, 1992.

[11] P. Gawthrop, "Selt tuning PID controllers : Algorithms and Implementation", IEEE transactions on automatic control, Marzo 1997.

[12] J. Gerry, "A comparison of PID control algorithms", Mayo 1997.

[13] Bollinger, John G., Duffie, Neil A.. "Computer Control of Machines and Processes", Addison-Wesley, USA, 1988.

[14] Everett, H.R. y Peters, A K. "Sensors for Mobile Robots" , Wellesley, MA,1995.

[15] Adams. M.D. “Sensor Modelling, Design and Data Processing for Autonomous Navigation.” World cientific publishing , Series in Robotics and Intelligent Systems. Singapore. Vol. 13. 1999.

[16] Olier Ivan Hernandez. Diseño e Implementación del Sistema de Control de un Robot Industrial.

[17] Stefans Spannare, "PWM drives for electric DC motor", Julio 2012.

[18] Microchip. "Datasheet dsPIC30F4011, Section 10. PWM", pp 4-10. 2005.

[19] Charles H. Lehmann, “Geometría Analítica”, pp 60-62. 1980.

[20] Charles H. Lehmann, “Geometría Analítica”, pp 80-84. 1980.

[21] Bollinger, John G., Duffie, Neil A. “Computer Control of Machines and Processes”, Addison-Wesley, USA, 1988.

[22] Ana Luiza de Almeida Pereira Zuquim, "An Embedded Converter from RS232 to Universal Serial Bus", IEEE

[23] Microchip. "Datasheet dsPIC30F4011, Section 19. UART", pp 3-7. 2004.

[24] Microchip. "Datasheet dsPIC30F4011, Section 11. I/O", pp 3. 2005.

[25] Microchip. "Datasheet dsPIC30F4011, Section 12. Timers", pp 2-8. 2005.

[26] Deitel, "Como programar en Java”, séptima edición, pag. 8.

[27] Microchip, "dsPIC Digital Signal Controlers", Octubre 2005.

[28] L5973D Datasheet, "Switching Regulator", Mayo 2002.

[29] L6203 Datasheet, "DMOS Full Bridge driver", Julio 1997.

Anexo A
Componentes Electrónicos

A.1 dsPIC30F4011

El dsPIC es un controlador digital de señales (DSC), el cual es un dispositivo programable que integra a la perfección los atributos de control de un microcontrolador (MCU) con las capacidades de cálculo y el rendimiento de un procesador digital de señales (DSP) en un solo núcleo [27] como se observa en la figura 7.1.

El dsPIC de Microchip posee las características más funcionales de un microcontrolador de 16 bits; manejo de interrupciones; una amplia gama de funciones digitales y analógicas; ahorro de energía; watch dog timer; seguridad de código; full-speed real-time emulation; etc.

Adicionalmente cuenta con la capacidad de un DSP para un alto rendimiento en operaciones matemáticas y operaciones de cálculo, así como en algoritmos de alto procesamiento de datos.

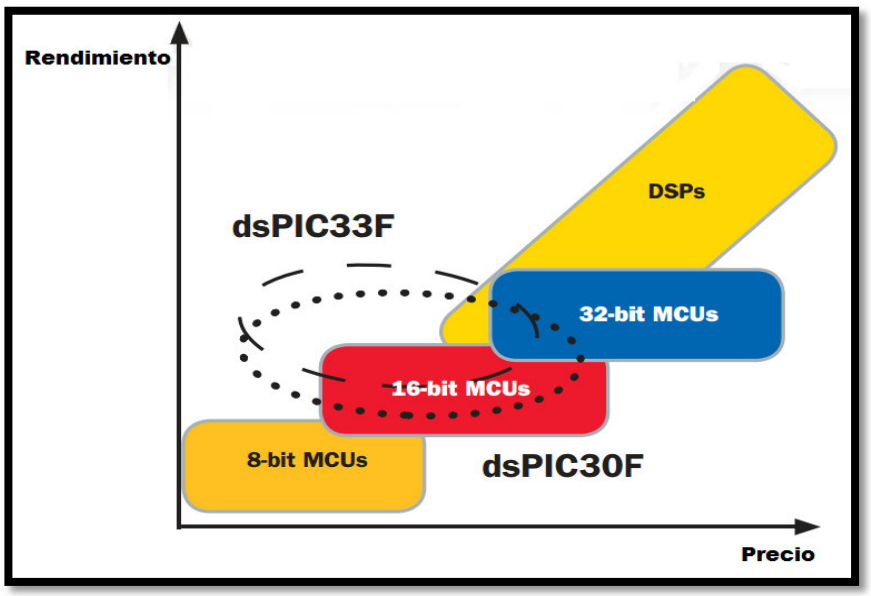


Figura 7.1.- Posicionamiento del dsPIC.

El dsPIC posee las mejores características de un MCU y un DSP, como se representa en la figura 7.2. El DSC es muy parecido a un microcontrolador MCU de 16 bits en cuanto a la arquitectura, repertorio de instrucciones y precio, pero con el rendimiento y las prestaciones de un DSP. Estos dispositivos se caracterizan por alcanzar un rendimiento de 40 MIPS e integrar memoria Flash de alta calidad junto a novedosos recursos de hardware, apoyándose en herramientas de desarrollo muy fáciles de manejar y manteniendo la compatibilidad de los diversos modelos con encapsulados de diferente patillaje.

Figura 7.2.- Unión de MCU y DSP.

La arquitectura (ver figura 7.3) de la CPU de los dsPIC30F se sustenta en un núcleo RISC con arquitectura Harvard mejorada. Actuando como soporte central de información posee un banco de 16 registros de 16 bits cada uno; se dispone de un bus de 16 líneas y otro de instrucciones de 24. Para potenciar la velocidad de las operaciones aritméticas complejas existe un "Motor DSP" que contiene un multiplicador de hardware rápido de 17x17 bits, dos acumuladores de 40 bits y un robusto registro de desplazamiento. La memoria de programa, tipo Flash, puede alcanzar un tamaño de 4M de instrucciones de 24 bits cada una. La memoria de datos se divide en dos espacios, X e Y, que pueden ser accedidos simultáneamente en las operaciones matemáticas del DSP. Toda esta estructura admite operaciones de MCU y de DSP en un repertorio de 84 instrucciones, la mayoría de 24 bits de longitud y ejecutables en un ciclo de instrucción.

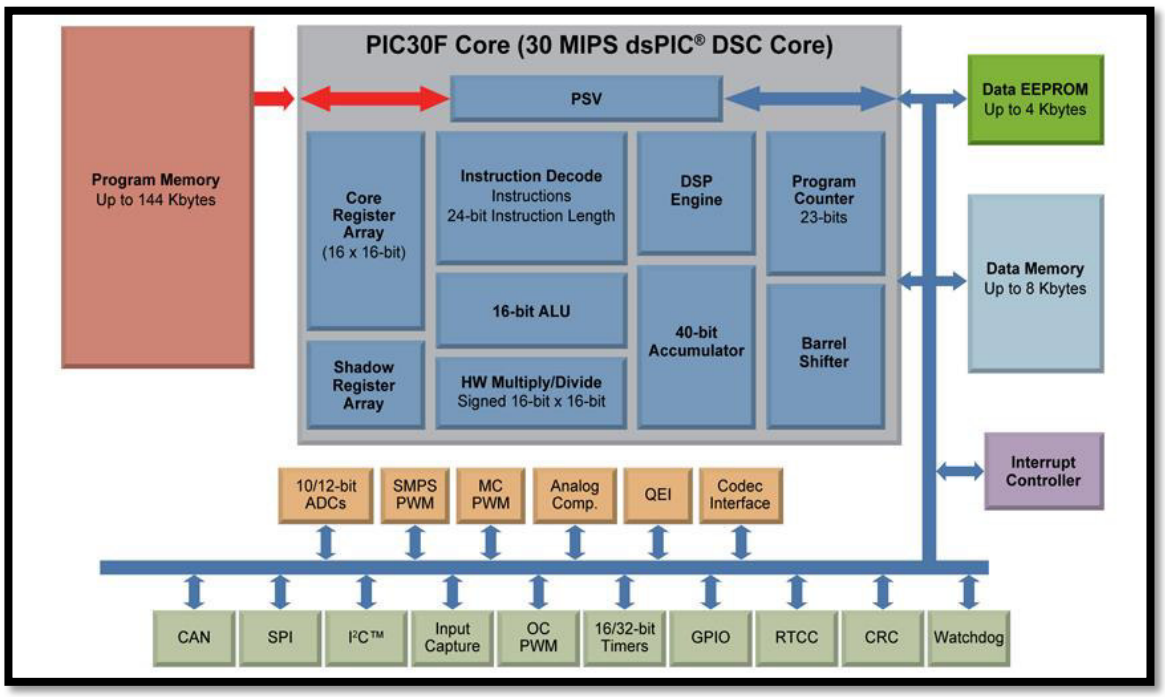


Figura 7.3.- Arquitectura de la familia dsPIC30F.

Las secciones MCU y DSP cooperan en el funcionamiento general y comparten el flujo de instrucciones de los DSC. Los recursos específicos del Motor DSP, además de soportar las operaciones DSP, permiten implementar nuevas y potentes instrucciones MCU. Para reducir el tiempo de ejecución de algunas instrucciones DSP la memoria de datos SRAM se divide en dos espacios X e Y que pueden ser accedidos a la vez.

Otra característica importante en los dsPIC30F es la de admitir hasta 45 fuentes distintas de petición de interrupción con 7 niveles de prioridad de los cuales 5 son externas.

Se puede encontrar con una gran variedad de periféricos (ver figura 7.4) como temporizadores, ADC, módulos de captura y comparación, módulos PWM para el control de motores, módulos de comunicación I2C, SPI, CAN, UART, etc. También disponen de potentes herramientas para la gestión del sistema Watchdog Timer, monitor de fallo de reloj, temporizadores para la estabilización de voltaje de alimentación y la frecuencia, etc. Así como dispositivos para controlar el consumo de energía.

Device	Pins	Program Mem. Bytes/ Instructions	SRAM Bytes	EEPROM Bytes	Timer 16-bit	Input Cap	Output Comp/Std PWM	Moto Control PWM	A/D 10-bit 500 Ksps	Quad Enc	UART	SPI™	I²C™	CAN
dsPIC30F2010	28	12K/4K	512	1024	3	4	2	6 ch	6 ch	Yes	1	1	1	-
dsPIC30F3010	28	24K/8K	1024	1024	5	4	2	6 ch	6 ch	Yes	1	1	1	-
dsPIC30F4012	28	48K/16K	2048	1024	5	4	2	6 ch	6 ch	Yes	1	1	1	1
dsPIC30F3011	40/44	24K/8K	1024	1024	5	4	4	6 ch	9 ch	Yes	2	1	1	-
dsPIC30F4011	40/44	48K/16K	2048	1024	5	4	4	6 ch	9 ch	Yes	2	1	1	1
dsPIC30F5015	64	66K/22K	2048	1024	5	4	4	8 ch	16 ch	Yes	1	2	1	1
dsPIC30F6010	80	144K/48K	8192	4096	5	8	8	8 ch	16 ch	Yes	2	2	1	2

Figura 7.4.- Características de la familia dsPIC30F.

El nivel de alimentación admite un voltaje de hasta 5.5V. Se tolera una temperatura interna entre -40º y 125º C y una externa entre -65º y 150º C, tal como se muestra en la figura 7.5. El rendimiento alcanza los 30 MPIS cuando el voltaje de alimentación tiene un valor de 4.5 y 5.5V DC.

Absolute Maximum Ratings

Ambient temperature under bias ..... -40°C to +125°C
Storage temperature ..... -65°C to +150°C
Voltage on any pin with respect to V_{SS} (except V_{DD} and $\overline{MCLR}$) **(Note 1)** ..... -0.3V to (V_{DD} + 0.3V)
Voltage on V_{DD} with respect to V_{SS} ..... -0.3V to +5.5V
Voltage on $\overline{MCLR}$ with respect to V_{SS} ..... 0V to +13.25V
Maximum current out of V_{SS} pin ..... 300 mA
Maximum current into V_{DD} pin ..... 250 mA
Input clamp current, I_{IK} (V_I < 0 or V_I > V_{DD}) ..... ±20 mA
Output clamp current, I_{OK} (V_O < 0 or V_O > V_{DD}) ..... ±20 mA
Maximum output current sunk by any I/O pin ..... 25 mA
Maximum output current sourced by any I/O pin ..... 25 mA
Maximum current sunk by all ports ..... 200 mA
Maximum current sourced by all ports ..... 200 mA

Figura 7.5.- Características de la familia dsPIC30F.

La nomenclatura de los chips de la familia dsPIC30F se muestra a continuación en la figura 7.6.

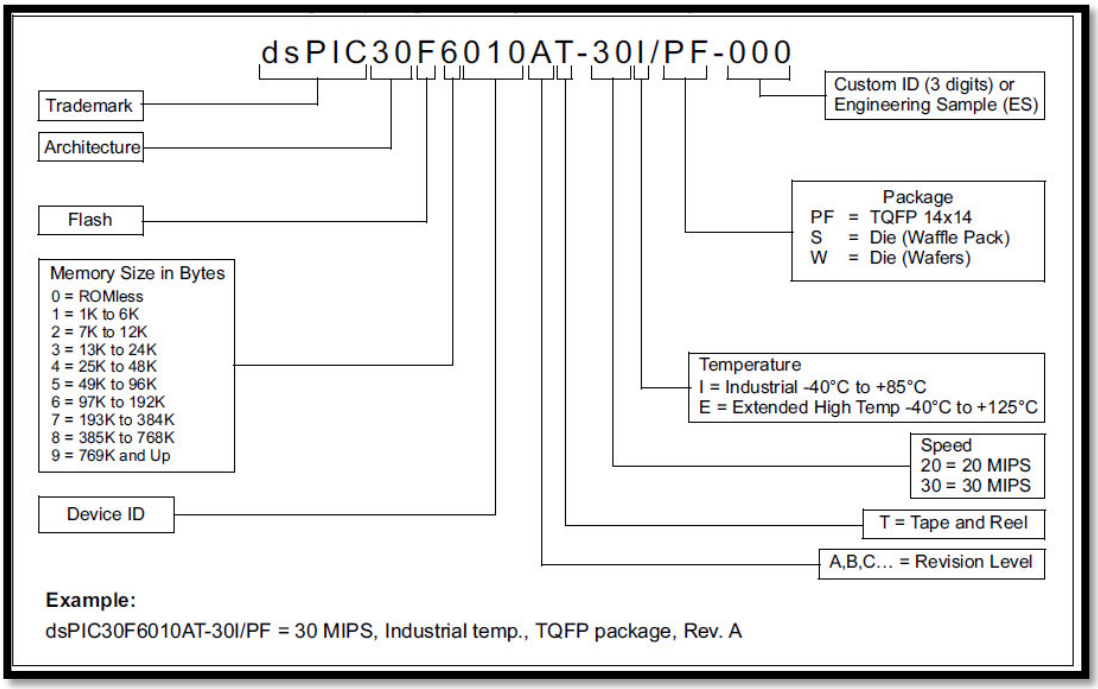


Figura 7.6.- Nomenclatura de la familia dsPIC30F.

Los dsPIC, así como los chips cuentan con distintos encapsulados. Para este proyecto se usará debido a sus dimensiones pequeñas y distribución de pines el encapsulado TQFP (ver figura 7.7).

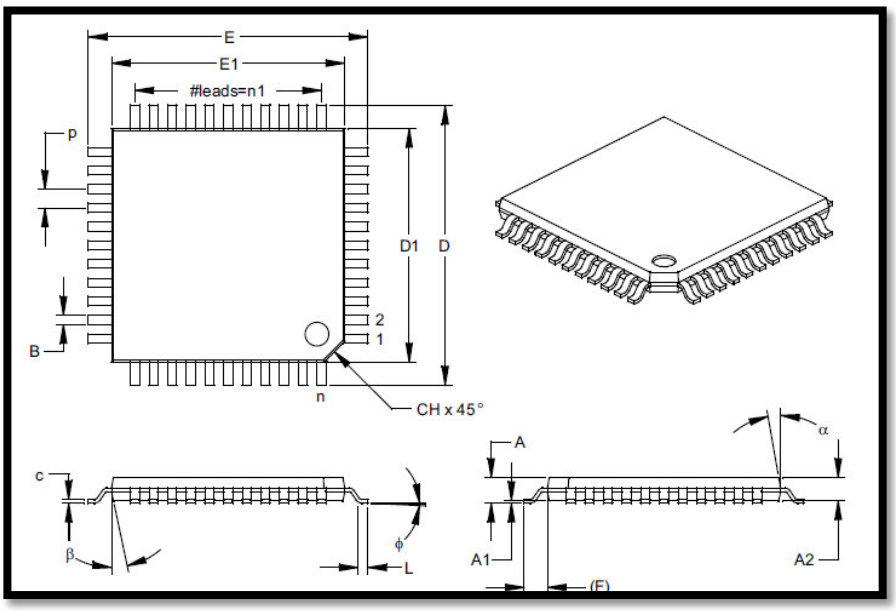


Figura 7.7.- Encapsulado TQFP del dsPIC30F4011.

Una de las ventajas de este encapsulado es que los pines están distribuidos alrededor del chip (ver figura 7.8), optimizando el espacio al momento de realizar el ruteo de la tarjeta electrónica.

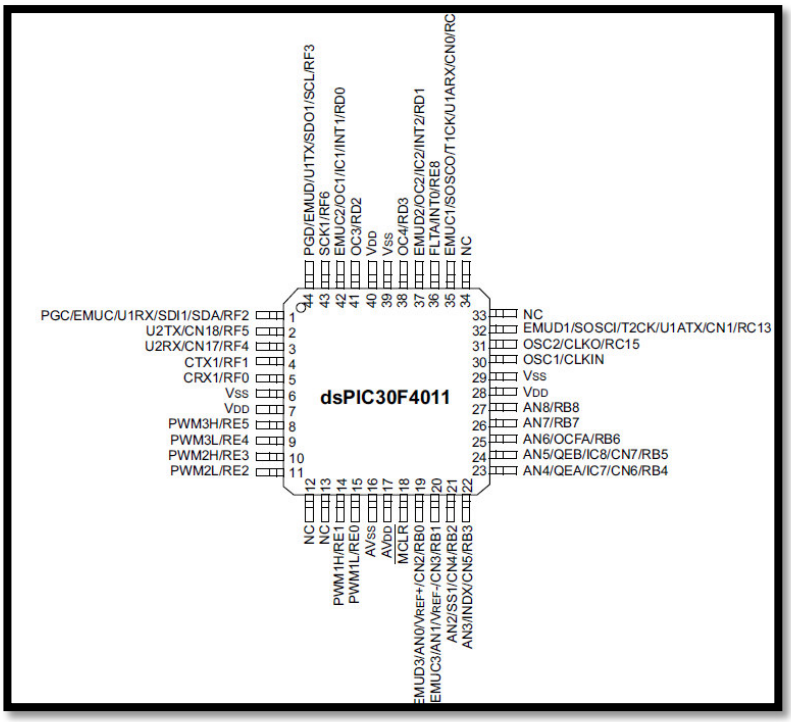


Figura 7.8.- Distribución de pines del dsPIC30F4011.

A.2 L5973D

El L5973D es un regulador switching pasa bajo, con una corriente límite de 2.5A y un voltaje de salida ajustable entre 1.0V hasta 40V [28]. El alto nivel de corriente se consigue gracias a un SO8 package con el marco expuesto, que permite reducir el Rth(j-amb) aproximadamente 40°C/W. El dispositivo utiliza un transistor D-MOS de canal P como elemento de conmutación para minimizar el tamaño de componentes externos y un oscilador interno que genera la frecuencia de conmutación a 250KHz.

La tensión de alimentación es de 4.4V hasta 40V (ver Figura 7.9); posee una referencia de salida de 3.3V; cuenta con un circuito de protección contra corto circuito; presenta un sistema de retroalimentación que facilita alcanzar una eficiencia del 90%; etc.

Symbol	Parameter	Value	Unit
V_8	Input Voltage	40	V
V_1	Output DC voltage Output peak voltage at t = 0.1µs	-1 to 40 -5 to 40	V V
I_1	Maximum output current	int. limit.	
V_4, V_5	Analog pins	4	V
V_3	INH	-0.3V to V_{CC}	
V_2	SYNC	-0.3 to 4	V
P_{tot}	Power dissipation at $T_{amb} \leq 60°C$	2.25	W
T_j	Operating junction temperature range	-40 to 150	°C
T_{stg}	Storage temperature range	-55 to 150	°C

Figura 7.9.- Características eléctricas del L5973D.

Este regulador cuenta con el package 8-SOIC (0.154", 3.90mm Width) ExposedPad (ver figura 7.10), que reduce el espacio dentro de la tarjeta electrónica.

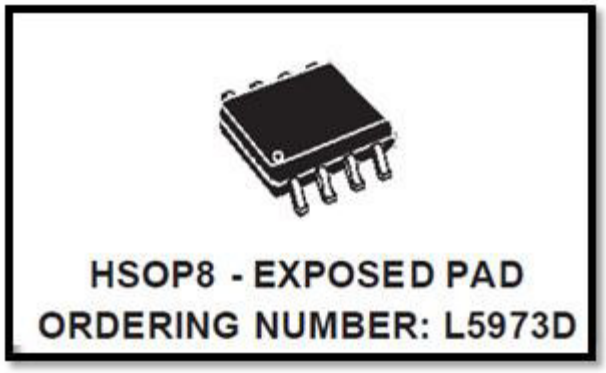


Figura 7.10.- Package del L5973D.

El L5973D cuenta con 8 pines (ver figura 7.11) distribuidos a los costados del chip. Las funciones de cada pin se pueden observar en la Tabla 7.1.

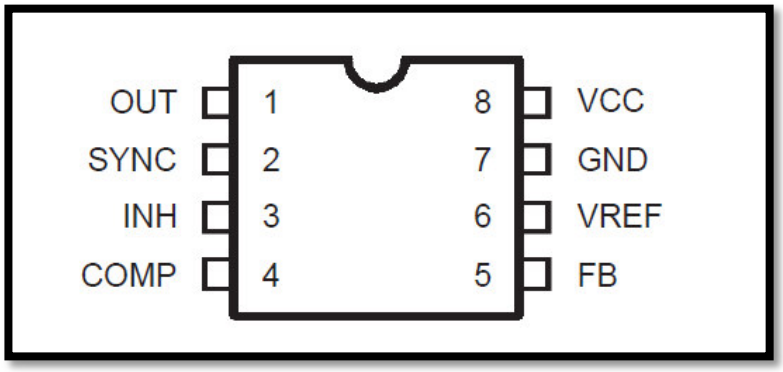


Figura 7.11.- Distribución de pines del L5973D.

Pin	Nombre	Descripción
1	**OUT**	Salida del regulador
2	**SYNC**	Pin de sincronización
3	**INH**	Una señal lógica deshabilita el dispositivo. Con INH mas alto que 2.2v el dispositivo está apagado y con INH más bajo que 0.8v el dispositivo esta prendido. Si INH no es usado, el pin debe ir a tierra.
4	**COMP**	Usado para compensar la frecuencia.
5	**FB**	Si conectamos el voltaje de salida directamente a este pin, el voltaje de salida resultante es de 1.235V. Para voltajes de salida más altos se requiere de un divisor externo resistivo.
6	**V_{REF}**	Voltaje de referencia de 3.3V.
7	**GND**	Tierra
8	**V_{CC}**	Voltaje DC de entrada

Tabla 7.1.- Funciones de los pines del L5973D.

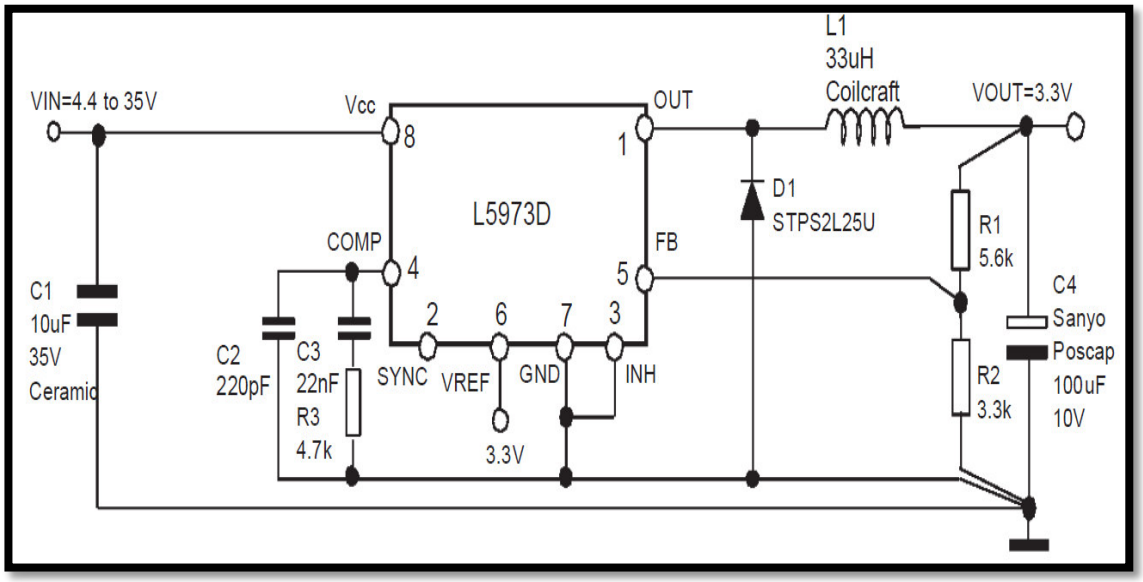


Figura 7.12.- Configuración del L5973D.

Para regular la tensión de salida a 5V utilizaremos la configuración mostrada en la figura 7.12, en esta configuración se utiliza el pin 5 para definir el valor de salida.

$$V_{OUT} = \frac{R1 + R2}{R2} x V_{FB} \quad \ldots (7.1)$$

Donde R1 es la resistencia entre salida (pin 1) y FB (pin 8); y R2 la resistencia entre FB y tierra. Según la tabla 7.1 el valor de V_{FB} es 1.235V y el voltaje V_{out} deseado

es 5V. Reemplazando estos valores en la ecuación 7.1, obtenemos una relación entre R1 y R2.

$$3.0485 \times R2 = R1 \ \ldots (7.2)$$

Se escogen valores de resistencia comerciales que se aproximen a la proporción, 7.5KΩ para R1 y 2.5KΩ para R2.

A.3 L6203

El L6203 es un driver puente H completo, para aplicaciones de control de motores en tecnología Multipower-BCD que combina transistores DMOS de potencia con circuitos CMOS y bipolares en el mismo chip [29]. Mediante el uso de tecnología mixta ha sido posible optimizar los circuitos de la lógica y la etapa de potencia para lograr el mejor rendimiento posible. Los DMOS de salida pueden funcionar a voltajes de hasta 42V y eficientemente conmutar a alta velocidad. Todas las entradas lógicas son TTL, CMOS y mC. Cada canal (medio puente) del dispositivo es controlado por una entrada lógica separada, mientras que el enable es compartido (ver figura 7.13).

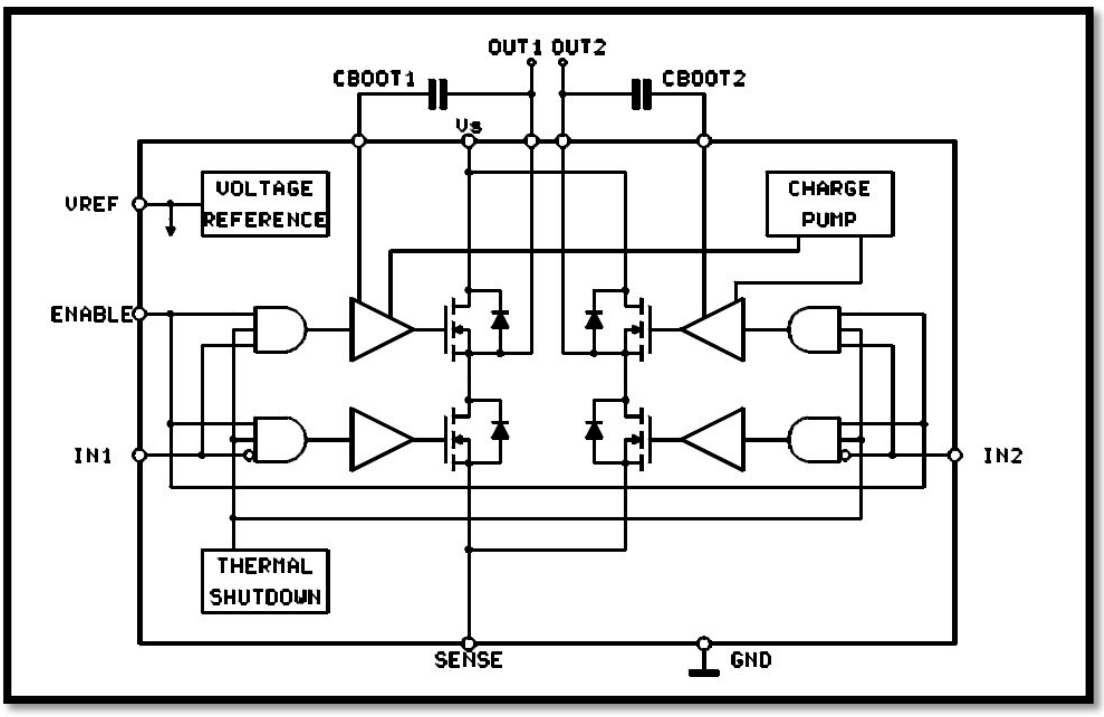


Figura 7.13.- Diagrama del L6203.

Este chip presenta las siguientes características (ver figura 7.14):

- Voltaje de alimentación de hasta con 48V.
- 5A de corriente de pico máxima.
- 4A de corriente RMS.
- $R_{DS\ (ON)}$ 0.3 Ω.
- Protección contra corto circuito.
- Frecuencia de operación hasta de 100KHz.
- Alimentación lógica interna.
- Alta eficiencia.

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_s	Supply Voltage		12	36	48	V
V_{ref}	Reference Voltage	I_{REF} = 2mA		13.5		V
I_{REF}	Output Current				2	mA
I_s	Quiescent Supply Current	EN = H V_{IN} = L EN = H V_{IN} = H I_L = 0 EN = L (Fig. 1,2,3)		10 10 8	15 15 15	mA mA mA
f_c	Commutation Frequency (*)			30	100	KHz
T_j	Thermal Shutdown			150		°C
T_d	Dead Time Protection			100		ns

Figura 7.14.- Características eléctricas del L6203.

El L6203 cuenta con distintos package, siendo el de mayor eficiencia y robustez el Multiwatt11 (ver figura 7.15).

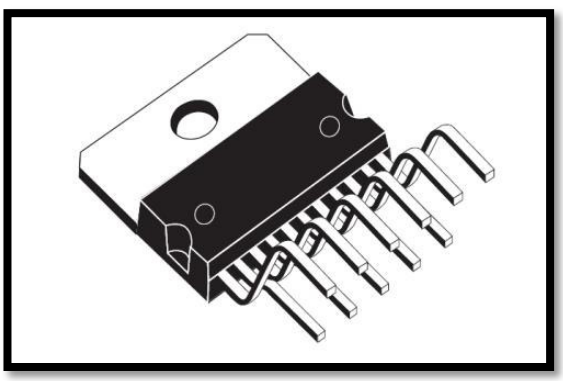


Figura 1.15.- Package Multiwatt11 del L6203.

El L6203 cuenta con 11 pines (ver figura 7.16) distribuidos debajo del chip. Las funciones de cada pin se pueden observar en la Tabla 7.2.

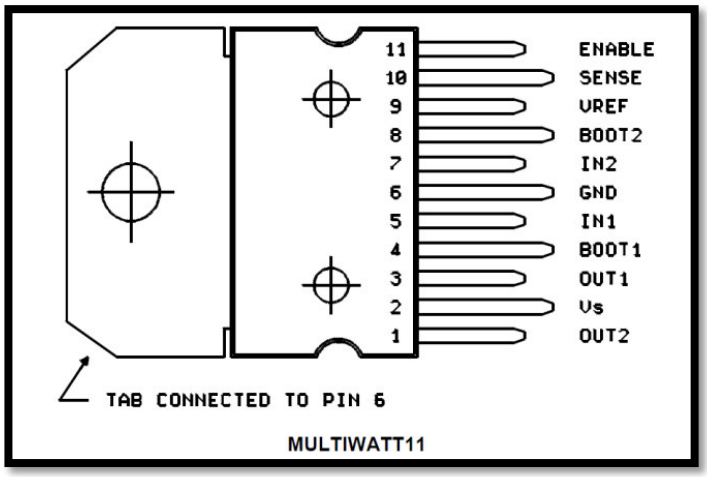


Figura 1.16.- Distribución de pines del L6203 - Multiwatt11.

Pin	Nombre	Descripción
1	**OUT2**	Salida de la segunda mitad del puente
2	**V_S**	Fuente de voltaje
3	**OUT1**	Salida de la primera mitad del puente
4	**BOOT1**	Un capacitor boostrap conectado a este pin asegura un funcionamiento eficiente del transistor DMOS superior
5	**IN1**	Entrada digital del controlador del motor
6	**GND**	Tierra común
7	**IN2**	Entrada digital del controlador del motor
8	**BOOT2**	Un capacitor boostrap conectado a este pin asegura un funcionamiento eficiente del transistor DMOS superior
9	**V_{ref}**	Voltaje interno de referencia. Se recomienda usar un capacitor entre este pin y tierra.
10	**SENSE**	Una resistencia R_{sense} conectada a este pin proporciona retroalimentación que nos permite controlar la corriente del motor
11	**ENABLE**	Cuando se tiene una entrada lógica alta en este pin los transistores DMOS son habilitados para ser manejados selectivamente por IN1 e IN2

Tabla 7.2.- Funciones de los pines del L6203 - Multiwatt11.

Para el control de un motor DC se utiliza la configuración mostrada en la figura 7.17. donde es necesario colocar los diodos D1 y D2 para evitar que las corriente regenerativas que se generan cuando el motor se desenergiza y el rotor sigue en movimiento comportándose como generador e induciendo una corriente en sentido contrario.

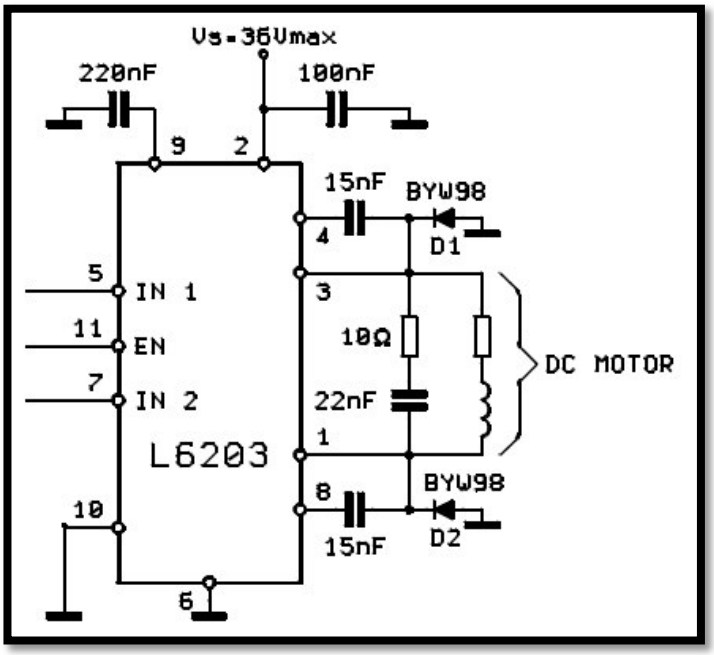


Figura 7.17.- Configuración del L6203 para control de motor DC.